UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA

TESIS DE MÁSTER

MÁSTER UNIVERSITARIO EN SOFTWARE Y SISTEMAS

Adaptación del comportamiento de agentes cognitivos mediante un modelo de inferencia de características definitorias de otros agentes

Autor: Nelly Victoria Salazar Mori

Director: Dr. Ricardo Imbert Paredes

Madrid, Julio de 2013

A mis padres, a Jorge, Eduardo y Daniel.

Agradecimientos

La realización de este trabajo sólo ha sido posible gracias a la intervención de personas muy valiosas que con su contribución han hecho que este proyecto se haga realidad.

Gracias, en primer lugar, a mi director Ricardo Imbert, cuya brillante idea ha dado origen de esta investigación. Además de ello, su paciencia, su capacidad para resolver los obstáculos encontrados y la dedicación de su valioso y, escaso, tiempo han sido admirables y dignos de un muy sincero agradecimiento.

Agradecer a mi familia, que ha accedido a tenerme tan lejos para realizar este Máster, y que semana a semana me demuestran un inagotable interés por este proyecto, brindándome ideas desde diversos puntos de vista que lo han enriquecido enormemente.

Finalmente, quiero agradecer a Dani por estar a mi lado en los momentos más duros de la realización de este trabajo, dándome la fuerza para superar el desánimo y restableciendo mi confianza.

Resumen

Las aproximaciones basadas en agentes se han convertido en una opción muy atractiva para muchos sistemas debido a las especiales características de este paradigma. Muchas de estas aplicaciones hacen uso de los agentes para estudiar o simular el comportamiento de los seres humanos en diversos contextos lo cual hace necesario la introducción de componentes determinantes en las personas como los rasgos de personalidad, las emociones, las actitudes, etc., que en su conjunto se ha denominado modelo personal. El objetivo es conseguir que el comportamiento de los agentes sea cada vez más creíble y supere las barreras generadas por considerar sólo los aspectos racionales. Así mismo, a medida que se produce la interacción entre personas, cada una de ellas va aprendiendo el modo de ser de las demás y va actualizando sus percepciones acerca de ellos, que serán de vital importancia para la toma de ciertas decisiones en las que necesite saber cómo es el otro. Es por ello que en este trabajo se presenta un modelo que permita a los agentes "conocer" el modelo personal de otros agentes basándose en la interacción entre ellos. Este modelo estará compuesto por un algoritmo genético que permita guiar ese aprendizaje. Finalmente se aplicará el modelo a un caso de estudio basado en un juego para comprobar el funcionamiento del mismo.

Palabras clave: Agentes virtuales, arquitecturas cognitivas, modelo personal, algoritmos genéticos.

Abstract

Agent-based approaches have become a very attractive option for many systems due to the special characteristics of this paradigm. Many of these applications use agents to study or simulate the human behavior in wide variety of contexts generating the need to introduce determining components of people as personality traits, emotions, attitudes, etc., which taken together has been called personal model. The goal is to get agent's behavior is increasingly credible and overcome the barriers imposed by taking into account only rational aspects. Moreover, as interaction between people occurs, each one is learning the way o being of the others and updating its perceptions about them, this is vitally important for making certain decisions in which it's necessary to know how the other is. Consequently, this paper presents a model, called inference model, that allows agents "to know" the personal model of the other agents based on the interaction between them. This inference model is composed of a genetic algorithm which will guide the agent's learning. Finally, the proposed model will be applied to a case study to test how it works.

Key words: Virtual agents, cognitive architecture, personal model, genetic algorithms.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

El deseo de crear entidades artificiales de forma y características humanas es una inquietud sorprendentemente antigua y que no sólo se ha manifestado en la comunidad científica, sino que también ha trascendido al mundo cinematográfico en el que sí "hacían realidad" que las máquinas tengan formas físicas humanas e incluso puedan adquirir razonamientos y sentimientos característicos de las personas. Sin embargo, en el mundo real este panorama no es tan alentador debido a la enorme complejidad que presentan los seres vivos que hace inviable la tarea de representarlos en un modelo que abarque todas sus características, lo cual ha conducido a los investigadores a modelar aspectos puntuales e ir enriqueciéndolos de poco a poco introduciendo variables representativas para el sector en el que serán aplicados. Dejando un poco de lado el "sueño" de construir humanoides que puedan comportarse como los seres humanos, una de las áreas que ha tenido grandes avances en este aspecto, es la de los entornos virtuales. En ésta se han conseguido realizar simulaciones cada vez más cercanas a las situaciones reales convirtiéndola en una potente herramienta aplicable en diversas áreas como educación, salud, entretenimiento, seguridad, etc.

El éxito de las simulaciones se encuentra en la capacidad de hacer que el entorno virtual y los elementos que interactúan en él se acerquen lo mayor posible a la situación real. Para ello, los diseñadores deben tener en cuenta las configuraciones más relevantes y de mayor impacto en estas situaciones de modo que puedan representarlas virtualmente para que los resultados sean aceptables de acuerdo con los objetivos de dicha simulación. En problemas que requieran variables o procedimientos que puedan ser medidos, tales como la temperatura, afluencia de personas, particularidades físicas de cada una, etc., éstos pueden ser calculados e implementados en una situación virtual consiguiendo que ésta sea bastante fiel a la realidad y pueda ser empleada con un grado de confiabilidad muy alto. Sin embargo, no en todo lo que se desea simular es posible medir todos los aspectos relevantes para una situación real. Un claro ejemplo de ello son las simulaciones en las que intervienen personas y que, por alguna razón, su comportamiento se ve influenciado por componentes propios de los seres humanos y, por qué no decirlo, animales, como son las emociones, las actitudes, los intereses, los rasgos de personalidad, etc., que forman parte del componente cognitivo y que se denomina el modelo personal. Ante ello, han surgido diversas arquitecturas basadas en agentes cognitivos

en los que su comportamiento esté acorde con su modelo personal haciéndolo cada vez más creíble y similar al modo de actuar de las personas. Una de las arquitecturas más significativas en este entorno es COGNITIVA ya que consigue integrar todos los componentes del modelo personal sin llegar a ser específica para ciertos problemas, ni demasiado genérica que no pueda ser aplicada a ninguno. Aunque con COGNITIVA se consigue que los agentes actúen guiados por su modelo personal, produciendo que bajo las mismas condiciones los agentes se comporten de forma distinta; aún queda por desarrollar un modelo para que un agente pueda intuir la "forma de ser" de los demás agentes con los que interactúa de modo que adapte su comportamiento y consiga maximizar su beneficio.

El presente trabajo consiste en mostrar que conociendo el modelo personal de los demás agentes, se consigue adaptar mejor el comportamiento propio para conseguir mejores resultados. Para ello, se propone el uso de un modelo de inferencia basado en algoritmos genéticos que será integrado a los procesos del agente para ser utilizado en la toma de decisiones en las que el hecho de "conocer" a los demás juegue un papel importante. El resultado de esta investigación es una primera aproximación de cómo dotar a los agentes virtuales de otra de las capacidades humanas de mayor relevancia en su comportamiento. Como parte de la solución propuesta, se plantean cuatro escenarios en los que se trata de cubrir todas situaciones en las que se puede conocer el modelo personal de los demás aunque sólo ha sido posible implementar y probar la primera de ellas por limitaciones temporales. En este primer escenario se tomará en cuenta sólo tres componentes del modelo personal que son (i) el conjunto de rasgos de personalidad o características definitorias, (ii) el conjunto de emociones y (iii) las relaciones entre ellos asumiendo que serán los mismos para todos los agentes. Dicho esto, la tarea se reduce a encontrar los valores de cada uno de los rasgos de personalidad de los demás agentes de modo que pueda intuir su comportamiento y tomar decisiones basadas en lo que cree que hará el otro. Se hace uso de algoritmos genéticos debido a su gran potencialidad cuando no se cuenta con conocimiento previo sobre la solución más adecuada, que es similar a cuando dos personas se conocen, y que va adaptándose progresivamente a medida que se va teniendo más información que se desprende del comportamiento de los demás.

El uso de este modelo podría enriquecer aún más las simulaciones en entornos virtuales haciendo que el comportamiento de los agentes tenga aún más credibilidad y consiga adaptarse al "modo de ser" de los demás consiguiendo, por ejemplo, que cuando se solicita algo a alguien y se obtiene una respuesta negativa, no se vuelva a solicitar lo mismo a ese mismo individuo repetidas veces; si no que más bien se busque a alguien que parezca pueda responder de forma positiva. Además de ello, con este trabajo se deja abierto el camino a realizar otras investigaciones relacionadas, contribuyendo así con el entorno académico ya que éste había sido un problema sin solución conocida a día de hoy.

Este trabajo está dividido siete capítulos, incluido el presente introductorio, en los que detalla el problema, la solución y un caso de estudio, finalizando con las conclusiones que han podido ser obtenidas y las líneas de trabajo futuro.

En el **capítulo 2** se realiza una descripción de los conceptos teóricos que serán empleados en el modelo que se desarrollará. El primero de ellos es el de Agente y se comienza hablando un

poco de su origen y estableciendo una definición, de las muchas que existen, que mejor aplique a este proyecto; se continúa con las diversas arquitecturas de agentes existentes y se finaliza con un apartado dedicado a los "agentes cognitivos" haciendo un resumen de la arquitectura COGNITIVA. El segundo concepto relevante para este trabajo es el de Algoritmos Genéticos y del mismo modo que para el anterior, se empieza dando a conocer su origen y algunas definiciones, para luego describir y relacionar la terminología que emplea con los términos propios del proceso evolutivo y concluir describiendo cada uno de los operadores genéticos que son aplicados cuando se implementa un algoritmo genético. Este capítulo concluye con la descripción de algunos proyectos relacionados, en los que se ha hecho uso tanto de agentes y algoritmos genéticos.

El **capítulo 3** está dedicado a detallar el planteamiento del problema, en el que se abordará una contextualización del mismo obteniendo la pregunta de investigación, así como también se darán a conocer los objetivo y la justificación de este trabajo.

En el **capítulo 4** se describe los escenarios identificados como parte de la solución y se establece el alcance del presente trabajo. Además, se explica cómo se llevará a cabo dicha solución, cuáles son los componentes de la misma y cómo puede ser validada.

El **capítulo 5** ofrece un caso de estudio para la validación del modelo propuesto. Se realiza las concreciones funcional y contextual acorde con COGNITIVA, se detalla la implementación de los agentes en JADE y la configuración del algoritmo genético finalizando con el análisis de los resultados obtenidos.

El **capítulo 6** corresponde al de las conclusiones que serán formuladas teniendo como base los objetivos planteados y, por último, en el **capítulo 7** se presentan las líneas de trabajo futuro desprendidas de este trabajo y que no han podido ser realizadas por las limitaciones encontradas.

Finalmente, se presentan algunos anexos con el código fuente del programa y una descripción de la plataforma para el desarrollo de sistemas basada en agentes, JADE.

Capítulo 2

Estado de la Cuestión

Este apartado está dedicado a dar una visión global de los temas relacionados con el presente trabajo.

2.1 Agentes

El término “agente” puede ser entendido de diversas formas, es así, que si se habla de agentes en un hospital sería muy probable que se refieran a agentes patógenos que distan mucho por ejemplo de un agente inmobiliario, término muy empleado en el sector de la compra y venta de inmuebles, o de un agente oxidante que se emplea en Química para oxidar otras sustancias. Debido a que al mencionar este término no se marca una contextualización implícita, es necesario, antes de continuar con el desarrollo del estado de la cuestión de agentes, especificar qué tipos de agentes se encuentran dentro del ámbito del presente trabajo.

Desde un punto de vista muy genérico, los agentes que se emplearán en el planteamiento de la solución y por ende los que se encuentran dentro del alcance de este documento son los agentes computacionales a los que, desde sus orígenes hasta hoy, se les ha ido atribuyendo diferentes adjetivos calificativos dependiendo de su funcionalidad o campo de investigación en el que fueron concebidos y desarrollados; así mismo se mencionará el término “agencia” que debe ser entendido como el área o campo que trata de los agentes. En los siguientes apartados se presenta una aproximación del estado de la cuestión de agentes.

2.1.1 Origen de la Agencia

Al iniciar el presente trabajo, se esperaba encontrar un panorama menos incierto con respecto al origen de la agencia y a la propia definición de agente (que será tratado en el siguiente apartado); sin embargo, éste no ha sido el caso. Se han publicado diversos documentos producto de innumerables investigaciones pero no todos convergen en la propiedad y origen de la agencia.

Nwana, por ejemplo, en su artículo “Software Agents: An Overview” publicado en 1996 menciona que el concepto de agente se remonta a la década de los 70’s cuando Carl Hewitt propone su “Modelo de Actor”, que es un modelo matemático que trata a los “actores” como

las primitivas universales de la computación digital concurrente (Hewitt, Bishop y Steiger, 1973; Hewitt, 1977). Hewitt define un actor como una entidad computacional que en respuesta a un mensaje que recibe, puede concurrentemente, enviar un mensaje a otros actores, crear nuevos actores o designar el comportamiento que se utilizará para el siguiente mensaje que reciba. Además menciona que no hay un orden supuesto para realizar estas acciones pudiendo ser llevadas a cabo de forma concurrente.

Si bien no se menciona explícitamente el término "agente" sino "actor", la característica que hace que pueda ser considerado como el origen de la agencia es la concurrencia, que de forma similar a los agentes, permite que puedan realizarse múltiples tareas simultáneas sin interferir unas con otras.

Es en 1986 que Marvin Minsky emplea el término "agente" en su teoría "Society of Mind", que pretendía dar respuesta a la interrogante de cómo funciona la mente humana, para referirse a la unidad mínima de lo que él llamó las sociedades de la mente. Un agente, entonces, representaba a cada componente de un proceso cognitivo que es suficientemente simple de entender. Además, se refiere a "agencia" para describir sociedades de agentes que en conjunto puedan realizar funciones más complejas en comparación de lo que puede hacer un agente de forma individual. Es más, un conjunto de agencias podían ser vistas como agentes unitarios si se ignora su composición interna y se considera solamente sus efectos externos. En este contexto, el concepto de agente tiene como antecesor a los sistemas de estructuras o *"Systems of frames"* con procedimientos adjuntos propuestas por Scott Fahlman, estudiante de Minsky en el año 1973 o 1974, para quien una estructura o *frame* era como un paquete de hechos relacionados y agencias, y que además podían incluir otras estructuras. Aunque esta teoría es algo más cercana, la comunidad científica no afirma que éste sea el origen de la agencia, aunque ha sido tomada como base para el desarrollo de investigaciones posteriores.

2.1.2 Definición de Agente

Hacia la década de los 90`s investigadores como Shohan, Nwana y Franklin comienzan a manifestar su malestar frente al uso del término "agente". Shohan, por ejemplo, que aunque el término agente se vuelve cada vez más popular, se ha usando de manera tan diversas que ha perdido sentido y no hace referencia a alguna noción particular de la agencia (Shohan, 1992).

Nwana, plantea dos líneas principales investigación en cuanto a agentes; la primera que abarca desde 1997 y cuyo objetivo fue de especificar, analizar, diseñar e integrar sistemas de múltiples agentes colaborativos y la segunda, a la que él se suma, que comienza alrededor de 1990 centrada en la investigación crítica de la diversificación de los tipos y clases de agentes ya que al parecer "todos" empezaron a denominar agente a "todo" (Nwana, 1996). Franklin, por su parte, menciona que se corre el riesgo de que cualquier programa sea denominado agente (Franklin, 1996). Este uso indiscriminado ha dado lugar a que se tengan múltiples definiciones que han desvirtuado este término y que, lejos de contribuir al desarrollo de los agentes, han hecho que se desvíe el hilo de investigación y, por ende, no se llegue a tener hasta el día de hoy una definición consensuada de lo que es un agente.

Otro aspecto a considerar es el gran número de áreas de investigación en las que se ha planteado el uso de agentes. Es así que Wooldridge menciona que diversos atributos asociados con la palabra agente son de importancia variable para diferentes dominios; por ejemplo en algunas aplicaciones, será más importante que los agentes aprendan de la experiencia y para otras no (Wooldridge, 1995).

Teniendo en cuenta lo anterior, en este trabajo no se pretende "descubrir" una definición que sea la panacea sino que se pretende encontrar alguna lo suficientemente genérica pero completa en la cual poder soportar la solución al problema de investigación planteado.

Al presentar tu trabajo sobre Programación Orientada a Agentes (AOP por sus siglas en Inglés), Shohan introduce el concepto de "estado mental" y plantea sistemas de alto nivel que emplean representaciones simbólicas, y que disfrutan, quizá, de algún tipo de función cognitiva (Imbert, 2005).

> *"Un agente es una entidad cuyo estado es visto como compuesto de componentes mentales tales como creencias, elecciones, aptitudes y compromisos." (Shohan, 1990)*

Stuart Russell y Peter Norvig, investigadores en el campo de la Inteligencia Artificial, no sólo dieron una definición de agente, sino que además mencionan lo que son los agentes racionales, ya que como ellos mismos lo mencionan, el tema central de su libro son los "agentes inteligentes" y éstos deben actuar de forma correcta.

> *"Un agente es cualquier cosa que pueda ver en su entorno a través de sensores y actuar en su entorno a través de efectores". (Russell & Norvig, 1995)*

> *"Un agente racional es el que hace lo correcto. Obviamente, esto es mejor que hacer las cosas mal, pero ¿qué significa? Como una primera aproximación, diremos que la acción correcta es la que hará que el agente sea más exitoso. Esto nos deja con el problema de decidir cómo y cuándo evaluar el éxito del agente." (Russell & Norvig, 1995)*

Esta definición no se limita a qué puedan ser los agentes: en los propios ejemplos que los autores proponen se citan a agentes humanos, para quienes los sensores sería los órganos de los sentidos y los efectores vendrían a ser las manos, piernas, la boca y otras partes del cuerpo; a agentes robots sustituyendo los sentidos por cámaras, por ejemplo, y teniendo motores como efectores e incluso se mencionan a los agentes de software que tengan cadenas de bits codificadas como percepciones y acciones.

Otra definición bastante general pero que limita a un agente a ser una pieza de software o hardware es la de Hyacinth S. Nwana que menciona que el término "agente" es una meta clase que abarca una amplia gama de otros tipos de agentes más específicos.

> *"El término agente se refiere a un componente de software y/o hardware que es capaz de actuar para poder ejecutar tareas en nombre de un usuario." (Nwana, 1996)*

A partir de la definición de Gilbert se hace una clara referencia a la necesidad de los agentes de ser autónomos y responder a su ambiente, posteriormente denominado entorno.

> *"Se consideran a los Agentes Inteligentes como una pieza de software que ejecuta una tarea dada utilizando información recolectada del ambiente, para actuar de manera apropiada hasta completar la tarea de manera exitosa. El software debe ser capaz de auto ajustarse basándose en los cambios que ocurren en su ambiente de forma tal que un cambio en las circunstancias producirá un resultado esperado." (Gilbert, 1995)*

La autonomía de los agentes se menciona explícitamente en la definición de Pattie Maes y va orientada al comportamiento de los agentes cuando realizan un conjunto de tareas. Otro aporte de Maes es que restringe el entorno de tal modo que deba ser complejo y dinámico.

> *"Un agente autónomo es un sistema computacional que habita en algún ambiente dinámico y complejo, percibiendo su estado y actuando autónomamente, llevando a cabo una serie de objetivos o tareas para los cuales fueron diseñados." (Maes, 1995)*

Stan Franklin y Art Graesser ponen énfasis en la contextualización, continuidad y retroalimentación del agente. Lo que distingue "agentes" de "programas ordinarios", mencionan, son la continuidad y la retroalimentación.

> *"Un agente autónomo es un sistema situado en, y parte de, un entorno que siente ese entorno y actúa sobre él, a través del tiempo, persiguiendo sus propios objetivos de forma que afecte lo que siente en el futuro." (Franklin, 1996)*

La Fundación de Agentes Físicos Inteligentes (FIPA por sus siglas en Inglés – Foundation for Intelligent Physical Agents), que es una organización para el desarrollo y establecimiento de estándares de software para sistemas basados en agentes, también ofrece una definición aunque muy ligada con su dominio.

> *"Un agente es una entidad de software encapsulado con su propio estado, conducta, hilo de control y la habilidad para interactuar y comunicarse con otras entidades (personas, otros agentes o sistemas legados)."*

Tal como se mencionó al principio de este apartado, se encuentra una larga y variopinta lista de definiciones de agente, y aunque no se cuestiona la validez de cada una en el área de investigación en la que se han desarrollado, se rescatan sólo la autonomía y que el agente se encuentre situado en un entorno. Es por ello que en este trabajo se tomará como base la definición propuesta por Wooldridge y Jennings (1995) que es la que goza de una mayor aceptación y es bastante concisa y completa. En una primera aproximación, los autores mencionan lo siguiente:

> *"Un agente es un sistema computacional que está situado en algún ambiente y que es capaz de actuar autónomamente en dicho ambiente con el fin de cumplir sus objetivos".*

Para luego complementar con sus características:

> *"El sentido más general en el que el término agente es usado es para denotar un sistema informático hardware o (más habitualmente) software que presenta las siguientes características:*
>
> *Autonomía: los agentes operan sin la intervención directa de los humanos u otros, y tienen algún tipo de control acerca de sus acciones y estado interno;*
>
> *Capacidad social: los agentes interactúan con otros agentes (y posiblemente humanos) por medio de algún tipo de lenguaje de comunicación de agentes;*
>
> *Reactividad: los agentes perciben su entorno (que puede ser el mundo físico, un usuario por medio de una interfaz de usuario gráfica, una colección de otros agentes, INTERNET, o quizá todos ellos combinados), y responden de un modo oportuno a los cambios que en él suceden;*
>
> *Proactividad: los agentes no actúan simplemente como respuesta al entorno, sino que son capaces de exhibir un comportamiento dirigido por metas tomando la iniciativa."*

Adicionalmente a estas características, se mencionan también las siguientes:

- continuidad temporal: los agentes son procesos continuamente en ejecución;
- adaptación o aprendizaje: los agentes cambian su comportamiento a partir de su experiencia previa;
- movilidad: los agentes son capaces de transportarse a sí mismos de una máquina a otra;
- flexibilidad: las acciones de los agentes se van adaptando a las condiciones del entorno;
- caracterización: los agentes pueden presentar personalidades creíbles y estados emocionales;
- veracidad: los agentes no comunican información falsa intencionalmente;
- benevolencia: los agentes no tienen objetivos contradictorios y siempre intentan realizar la tarea que se les solicita;
- racionalidad: los agentes tienen objetivos específicos y siempre intentan llevarlos a cabo.

Para complementar la definición de agente, es útil mencionar aquello que un agente no es, ya que, producto del uso indiscriminado del término, otros conceptos pueden llevar a confusión. La primera comparación que se han planteado diversos autores está orientada a diferenciar los agentes de los objetos (Programación Orientada a Objetos). Jennings, por ejemplo, considera a los agentes como los sucesores del paradigma de objetos; Wooldridge, por su parte dice *"objects do it for free, agents do it for Money"* y Parunak menciona que un agente es:

- Un objeto con iniciativa.
- Un objeto con actitud u orientación.
- Un objeto que puede decir "No" (o "Adelante").
- Un objeto proactivo.

Un objeto es una entidad compuesta por atributos, que determinan su estado, y métodos, que determinan su comportamiento, y que al ser invocados se ejecutarán indefectiblemente produciendo los resultados esperado. Ésta es la diferencia más marcada con los agentes, cuando éstos reciben la instrucción de ejecutar alguno de sus comportamientos, decidirán si les conviene o no hacerlo y caso positivo lo harán.

La segunda comparación que se encuentra en la literatura es con los Sistemas Expertos, que son un conjunto de programas que imitan las actividades de una persona para resolver problemas diversos y cuyo poder de resolución se basa en el conocimiento de un dominio en particular. Las diferencias básicas que se encuentra entre agentes y sistemas expertos son que estos últimos no interactúan directamente con el entorno, suelen diseñarse para tareas más complejas donde ellos tienen todo el control de las decisiones para ese dominio y no suelen cooperar entre sí.

2.1.3 Arquitecturas de Agentes

Desde una visión más abstracta, una arquitectura es una metodología general para diseñar una descomposición particular de una tarea en módulos típicamente representados por cajas con flechas que indican el flujo de control de datos entre los módulos (Kaebling, 1991). El conjunto total de módulos debe responder a la pregunta de cómo la data de los sensores y el estado interno del agente determinan sus acciones y el futuro estado interno del agente. Una arquitectura abarca las técnicas y los algoritmos que soportan esta metodología (Maes, 1991).

Antes de detallar las arquitecturas de agentes, se expondrán las dos nociones mencionadas por Wooldridge, ya que pueden ser consideradas como macro niveles de las arquitecturas existentes (Wooldridge, 1995):

La primera es la *noción débil*, que consiste en definir un agente como una entidad que es capaz de intercambiar mensajes utilizando un lenguaje de comunicación. Esta definición es la más utilizada dentro de la ingeniería del software basada en agentes. Su finalidad es conseguir la interoperabilidad entre aplicaciones a nivel semántico por medio de la emergente tecnología de agentes. Las características que debe tener un agente según esta noción son: autonomía, iniciativa y sociabilidad.

Es bajo esta noción, que se encuentra la clasificación de arquitecturas de agentes por el modelo de razonamiento (Imbert, 2005) o por el tipo de procesamiento (Iglesias, 1998), que puede considerarse la más conocida y que presenta las arquitecturas deliberativas, reactivas e híbridas.
Relacionados con este criterio de clasificación y con el modelo de capas, se presentan dos enfoques atendiendo al criterio de acceso a los sensores y actuadores de las capas de control. El primero de ellos considera que las capas se encuentren dispuestas horizontalmente, lo cual permite que todas estas capas tengan acceso tanto a la información de entrada como a la generación de salida. Sin embargo, con el segundo enfoque, sólo una de las capas tiene acceso a las entradas percibidas y, del mismo modo, sólo una de ellas deberá generar la salida.

La segunda corresponde a la *noción más fuerte* o reactiva de agentes. Fue enunciada por Shoham, en su propuesta de programación orientada a agentes (AOP), en la que un agente se

define como una entidad cuyo estado es visto como un conjunto de componentes mentales, tales como creencias, capacidades, decisiones y acuerdos (Shoham, 1193).
Esta noción, por su parte, presenta arquitecturas más particulares. Un ejemplo de estas arquitecturas son aquellas que están basadas en creencias, deseos e intenciones.

2.1.3.1 Arquitecturas según el criterio débil de agentes

Arquitecturas deliberativas

Este enfoque consiste en introducir una función deliberativa entre la percepción y la ejecución, con el objetivo de seleccionar el comportamiento correcto. Esta función requiere dos procesos: el primero se empleará para decidir qué objetivos perseguir, que viene a ser la deliberación; y el segundo será usado para determinar cómo alcanzar dichos objetivos, que se denomina razonamiento basado en medios y fines.

Internamente, los agentes estarán dotados de modelos de representación simbólica del conocimiento, definidos acorde con el problema de investigación, sobre lo cual el agente deberá razonar y llevar a cabo las tareas encomendadas en el tiempo preestablecido.

Esto está fuertemente relacionado con las teorías clásicas de planificación en Inteligencia Artificial. A partir de un modelo simbólico del universo, una especificación simbólica de las acciones disponibles y un algoritmo de planificación que manipule sintácticamente dichas representaciones, el agente es capaz de generar un plan para alcanzar un objetivo partiendo de un estado inicial (Imbert, 2005).

Algunos tipos de agentes deliberativos son: los agentes intencionales, que pueden razonar sobre sus creencias e intenciones; y los agentes sociales, que pueden ser vistos como cooperativos o como agentes intencionales, pero con un modelo explícito de otros agentes.

Aunque en muchos documentos se presentan las arquitecturas BDI como parte del enfoque deliberativo (debido a que éstas surgieron como arquitecturas paradigmáticamente deliberativas), con el paso del tiempo han dado lugar a implementaciones basadas en arquitecturas híbridas, por lo que se detallarán en el apartado correspondiente a la noción fuerte de la agencia.

El principal inconveniente que presenta este tipo de arquitecturas, es que requieren mucho tiempo para la evaluación de las entradas y el desarrollo de planes basadas en ellas. Esto supone un problema cuando el entorno es muy cambiante, ya que un plan o decisión que un agente puede tomar, podría ya no estar acorde con las condiciones del entorno y, por ende, daría una respuesta equivocada.

Arquitecturas Reactivas

Las arquitecturas reactivas no incluyen el modelo del mundo simbólico, presente en las deliberativas; por lo tanto, no son capaces de emplear razonamientos simbólicos complejos y, más bien, funcionan en relación al modelo estímulo-respuesta (en la que el estímulo proveniente de los sensores tiene un efecto directo sobre los actuadores). Surge como una alternativa al problema del tiempo de respuesta de las arquitecturas deliberativas.

Un ejemplo del enfoque reactivo es la arquitectura de subsunción propuesta por Roodney Brooks, quien menciona que las arquitecturas reactivas se caracterizan por no tener como elemento central de razonamiento un modelo simbólico y por no utilizar razonamiento simbólico complejo (Brooks, 1991).

Las principales arquitecturas reactivas creadas son: las reglas situadas, las arquitecturas de subsunción y autónomas de estado finito, las tareas competitivas y las redes neuronales.

Estas arquitecturas presentan dos inconvenientes principales: el primero es que son difíciles de diseñar si se busca que los agentes puedan aprender de la experiencia; y el segundo tiene relación con los agentes con múltiples comportamientos, haciendo difícil de entender sus interacciones.

Arquitecturas Híbridas

Las arquitecturas híbridas surgen debido a las carencias de las arquitecturas anteriores para resolver cualquier tipo de problema. Lo que pretende es tomar los beneficios y potencialidades de cada una de ellas, de modo que, dependiendo del dominio del problema, puedan diseñarse arquitecturas de diversos niveles.

Es común que el nivel inferior de las arquitecturas híbridas corresponda con el enfoque reactivo, de modo que, los agentes puedan responder en el tiempo adecuado a peticiones que no requieren una planificación de acciones en función de sus objetivos, que es tarea del segundo nivel de esta jerarquía, que hereda las características de la filosofía deliberativa.

Además de estos dos niveles, se requiere un nivel más de abstracción para dotar a los agentes de capacidades sociales, que les permita tomar decisiones en cuanto a la cooperación y competición con otros agentes. Teniendo como resultado que el comportamiento global del agente está definido por la interacción entre estos niveles.

No todas las arquitecturas híbridas deben tener los tres niveles propuestos. El diseño está fuertemente ligado al dominio del problema, que dependiendo del caso, requieran todos o algunos de estos niveles.

2.1.3.2 Arquitecturas según el criterio fuerte de agentes

Según la programación orientada a agentes, los agentes son entidades que presentan su estado como un conjunto de componentes mentales, dando origen a la noción fuerte de la agencia. Además de las características la agencia débil, requieren propiedades como creencias, deseos e intenciones, dando lugar a arquitecturas más particulares que serán tratadas en este apartado.

Arquitecturas BDI

BDI son las iniciales de *Belief, Desiere and Intention* y significa Creencias, Deseos e Intenciones. Como arquitectura de agentes está caracterizada por que los agentes están dotados de estados mentales (Haddadi y Sundermeyer, 1996), en donde las creencias representan el conocimiento del agente sobre el entorno, vital en medios dinámicos para mantener información sobre los eventos pasados y al mismo tiempo para permitir la adaptación y

evolución del agente; los deseos son las metas del agente y representan los estados finales deseados, que para ser alcanzados a partir de las creencias, es necesario definir un mecanismo de planificación que permita identificar las intenciones. Por último, las intenciones son un conjunto de caminos de ejecución que pueden ser interrumpidos, de una forma apropiada, al recibir información sobre cambios del entorno, y que manejan y conducen a las acciones hacia las metas de manera persistente e influyendo en las creencias.

Hasta este punto no es difícil darse cuenta de que estas características son, de manera muy básica, algunas de las que presenta el ser humano. Es por ello, que los agentes BDI permiten el desarrollo de sistemas que se integren adecuadamente en el mundo real (Georgeff, 1998). El tener en cuenta las propiedades de la persona en el diseño de agentes, nos conduce a lo que se ha denominado *razonamiento práctico,* que debe sopesar los pros y contras de consideraciones conflictivas que compiten entre sí, donde las opciones relevantes son proporcionadas por los deseos, valores, aspectos de importancia y creencias (Bratman, 1990). El razonamiento práctico consiste en dos actividades principales: la primera de ellas es la deliberación, que consiste en decidir cuál es el estado que se quiere alcanzar; y la segunda es el razonamiento medios-fines, que consiste en decidir cómo alcanzar ese estado. Mientras las salidas de esta segunda actividad son los planes, las salidas de la primera son las intenciones y juegan un importante papel en el razonamiento práctico. Dennett define un agente como un sistema intencional (Dennett, 1987) en el que las intenciones dirigen el razonamiento basado en medios y fines, restringen las deliberaciones futuras, persisten e influencian las creencias sobre las que se basará el futuro razonamiento práctico. Éstas deberán replantearse cada cierto tiempo para encontrarse acorde con el entorno.

Una arquitectura BDI está especialmente indicada para contextos con las siguientes características (Rao y Georgeff, 1995):

- Entorno no determinista.
- Sistema no determinista.
- Posible coexistencia simultánea de varios objetivos, los cuales pueden no ser satisfechas a la vez.
- Las acciones para alcanzar los objetivos propuestos son dependientes del entorno, pero independientes del estado del propio sistema (agente).
- Sólo se reciben percepciones locales del entorno.
- Pueden haber cambios en el entorno tanto durante la decisión de las acciones como durante su ejecución.

Un agente BDI consta de los siguientes componentes:

- Un conjunto de creencias actuales del agente sobre su entorno.
- Una función de revisión de creencias, que actualiza las creencias en función de las percepciones.
- Una función de generación de opciones (deseos) a partir de sus creencias e intenciones.
- Un conjunto de opciones actuales.
- Una función filtro para al proceso de deliberación, que determina las nuevas intenciones en base a sus creencias, deseos e intenciones.
- Un conjunto de intenciones actuales.

- Una función de selección de acciones que determina qué acción ejecutar a partir de las intenciones actuales.

El mecanismo básico de funcionamiento de una arquitectura BDI consiste en generar nuevas opciones en respuesta a los cambios en el entorno o nuevas necesidades. Estas opciones son filtradas, y las que permanecen después de este filtrado, son sometidas a un proceso de deliberación para producir intenciones que se van incorporando a los planes que el agente tiene.

2.1.4 Agentes Cognitivos

Hasta aquí se ha adoptado una definición y se han identificado las características y propiedades de los agentes en términos generales. Se ha dicho que los agentes tendrán comportamientos definidos que se adaptarán al entorno para conseguir sus metas. Estos comportamientos deberán ser definidos teniendo en cuenta a qué representarán los agentes. Esto es importante debido a que si se construyen agentes para máquinas, será relevante definir sus comportamientos en función de situaciones lógicas que los controlarán, y su toma de decisiones se hará tomando en cuenta, básicamente, los valores percibidos del entorno y aplicando diversos criterios de situaciones conocidas que podrán afinarse cada vez más para conseguir resultados cada vez más exactos.

Sin embargo, si lo que se pretende es representar agentes con características antropomórficas, no bastará con aplicar solamente situaciones racionales para tomar una decisión, ya que en ocasiones, y dependiendo del entorno, el resultado sería totalmente distinto al de una persona e, inclusive, en situaciones exactamente iguales del entorno, varias personas actuarían de modo completamente distinto unas de otras. Un ejemplo muy claro de ello es el comportamiento de un grupo de personas frente a un desastre natural: si se observara el panorama sin estar dentro de él, se podría ver que sólo algunos actuarán con cautela y siguiendo las normas de seguridad conocidas y/o ensayadas previamente, tratando de mantener la calma y el orden, que es lo más razonable y lo que se espera que se haga; otros, sin embargo, entrarán en pánico y echarán a correr intentando huir y encontrar un lugar que consideren seguro; otros caerán en un estado de impotencia y no realizarán ninguna acción pese a la necesidad intrínseca del ser humano de poner a salvo su vida, necesidad que también será pasada por alto por aquellos que se armarán de valor y pretenderán ayudar a otros, exponiéndose a peligros aun cuando ya estuvieran a salvo.

En el ejemplo anterior se ha mencionado las expresiones “mantener la calma”, “entrar en pánico”, “estado de impotencia”, “armarse de valor” que no son más que estados emocionales que se manifestarán en mayor o menor medida dependiendo de las “características” propias de cada persona que vienen dadas, básicamente, por su personalidad. Si sumamos además sus actitudes frente a los demás y sus intereses, se podrá cambiar radicalmente la decisión tomada.

Es por ello, que para conseguir que los agentes, definidos básicamente como sistemas informáticos, tengan capacidades cognitivas, se deberá tomar en cuenta todo este componente no racional compuesto por emociones, personalidad, actitudes, intereses, etc.

Si bien las definiciones de “agente cognitivo” no mencionan explícitamente el término “emoción”, el hecho de intentar simular comportamientos humanos hace que ésta debe ser incluida junto con otras características determinantes que no pueden apartarse de la naturaleza personal.

Se puede definir entonces a un agente cognitivo, como aquél que es capaz de razonar en función de lo que percibe y tomar decisiones sobre la mejor forma de actuar, de forma similar a como lo haría un ser humano. Poseen una estructura compleja que incluye los componentes que forman su estado mental, como son: creencias, conocimiento, comunicación, control y su funcionalidad.

De modo general, las características de un agente cognitivo puede ser las siguientes:

- Representación explícita del ambiente y de otros agentes.
- Memoria de acciones (histórico).
- Organización social.
- Mecanismo de control deliberativo.
- Comunicación directa entre los agentes.

2.1.4.1 Arquitectura Cognitiva del Agente

En este apartado se describe a COGNITIVA, arquitectura que fue propuesta por Ricardo Imbert en 2005 y plantea una arquitectura completa para módulo cognitivo de agentes con carácter emocional. El problema principal al que COGNITIVA pretende dar solución es la extrema particularización o generalización de las arquitecturas existentes en este contexto que según el caso, tienen que ser rediseñadas por completo cuando se pretende emplearlas en dominios distintos, o que no pueden aplicarse a algún problema del mundo real. Es así, que COGNITIVA divide las estructuras y funciones que presenta el contexto de aplicación consiguiendo con ello, que mediante un proceso de aplicación incremental pueda adaptarse a distintos entornos y problemáticas con una flexibilidad considerable (Imbert, 2005). A nivel arquitectónico, COGNITIVA presenta el modelo bastante extendido en el mundo de los agentes, *percepción-cognición-acción*. Éste emplea sensores para la captura de información de entrada y actuadores para realizar las acciones que han sido planificadas en el módulo cognitivo, que viene a ser el tema central desarrollado en la tesis doctoral de su creador.

El módulo cognitivo aprovecha las capacidades de las arquitecturas deliberativa y reactiva traduciéndose en una arquitectura híbrida, que como ya se vio en el apartado 2.1.3.1, se encuentran compuestas de niveles o capas y, según el problema, tendrán comportamientos reactivos, deliberativos y sociales. De acuerdo con ello, COGNITIVA presenta estos tres niveles en disposición horizontal, con respecto a la información recibida de los sensores y las acciones a realizar por los actuadores, tal como se puede observar en la Figura 1.

Además de estos tres niveles básicos, presenta también una capa *intérprete* cuyo objetivo es recoger, filtrar y traducir las percepciones de modo que puedan ser interpretadas por los niveles reactivo, deliberativo o social y otra capa *organizador*, encargada de ordenar y secuencias las acciones hacia los actuadores.

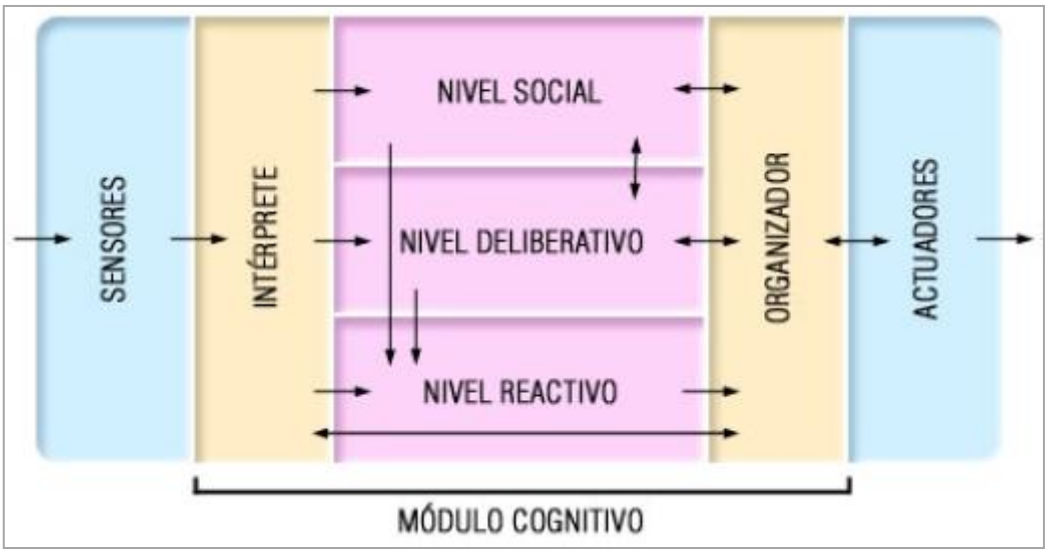

Figura 1 - Arquitectura general de COGNITIVA.

Los flujos básicos de comunicación entre todos los componentes de COGNITIVA son los siguientes (Imbert, 2005):

- el intérprete recibe continuamente, de los sensores, notificaciones de cambios de estado, eventos e información;
- los tres niveles –reactivo, deliberativo y social– del módulo cognitivo reciben a su vez, del intérprete, sólo aquella parte de toda esta información, eventos y cambios de estado que les resultará de interés;
- los niveles social y deliberativo procesan esa entrada y conforme a determinados factores, generan nuevas metas y proponen posibles planes para satisfacerlas, cada nivel desde su perspectiva particular;
- a su vez, de acuerdo con las metas trazadas, indican al nivel reactivo las pautas que debe seguir en su proceder, para que éste sea acorde a dichas metas;
- este nivel reactivo, a partir, fundamentalmente, de la entrada procesada por el intérprete y de las indicaciones recibidas, propone al organizador las acciones que considera necesarias;
- el organizador recoge los planes y acciones propuestos y los estructura para que no haya conflictos entre ellos, tratando de optimizar el comportamiento del agente;
- a partir de la *agenda* generada por el organizador, éste es capaz de ir proporcionando a los actuadores las acciones seleccionadas en el orden oportuno;
- por último, el organizador comunica al intérprete cuáles son las expectativas que se tienen de acuerdo con las acciones que se van ejecutando. Por su parte, el intérprete indica al organizador si las percepciones recibidas van siendo acordes a las expectativas manifestadas.

En las siguientes líneas se realiza una descripción de cada uno de estos niveles.

Nivel reactivo

La filosofía de un nivel reactivo en una arquitectura de agentes es básicamente la de responder rápidamente a los cambios que puedan producirse en el entorno o a nueva información que el agente perciba, enlazando las percepciones con las acciones. Esto encuentra su justificación, en que hay solicitudes que requieren respuesta en un tiempo tan reducido, que sería imposible ejecutar procesos deliberativos que cumplan en tiempo con ésta.

A diferencia del comportamiento de un agente puramente reactivo, en COGNITIVA, hablar de reacción no implica necesariamente suspensión momentánea del control racional (Imbert, 2005), teniendo en cuenta, además de la percepción, las creencias que el agente tiene que son esenciales para hacer creíble su comportamiento.

De forma similar a las personas, los agentes tendrán ciertas creencias para todo cuanto le rodea, o lo que percibe de lo que le rodea. En COGNITIVA, se toman en cuenta las creencias hacia los recintos, que vienen a ser los ambientes en sí donde el agente puede encontrarse, las creencias de los objetos, que existen en el entorno, las creencias de los individuos con los que el agente interactúa y las creencias sobre la situación actual, que a diferencia de las tres anteriores que son tangibles y tienen características propias, éstas depende más de las percepciones del propio agente.

En cuanto a las creencias tangibles, presentan en común tres componentes que son sus *características definitorios*, que cambiarán muy poco y muy lentamente a lo largo del tiempo, los *estados transitorios*, que son más dinámicos y marcarán un período máximo de validez, y las *actitudes* que despiertan en el agente, que son importantes en la toma de decisiones para la selección de acciones y generación de comportamientos coherentes. En la Tabla 1 se detalla la información, que por ejemplo, podría tener cada uno de estos componentes para los recintos, objetos e individuos.

Creencias	Características definitorias	Estados transitorios	Actitudes
Recintos	Identificación Mapa Posición Función	Número de individuos Temperatura	Confianza en un determinado lugar
Objetos	Identificación Función Tipo	Lugares donde podría encontrase el objeto	Temor hacia un objeto
Individuos	Identificación Rasgos de personalidad Habilidades Estados físicos	Emociones Rasgos físicos	Simpatía por otro agente

Tabla 1 – Información de ejemplo de recintos, objetos e individuos.

En cuanto a las creencias sobre los individuos, la combinación de las características definitorias y los estados transitorios de cada uno de ellos formarán su *modelo personal,* que determinará el comportamiento del individuo y hará que éste no sea igual al comportamiento de los demás en las mismas condiciones, e incluso sea diferente en situaciones iguales pero que ocurran en instantes distintos.

Debido a que uno de los componentes del modelo personal es dinámico, éste deberá actualizarse continuamente para seguir siendo coherente a medida que el tiempo avance y los cambios se vayan produciendo. Para ello COGNITIVA tiene en cuenta las relaciones entre las creencias tales como las emociones, personalidad, estados físicos y actitudes. En agentes emocionales, su comportamiento deberá estar fuertemente influenciado por la emoción que presente el agente en un momento dado, permitiendo explicar manifestaciones no siempre racionales pero que se producen con frecuencia en el mundo real. Las emociones, a su vez, aflorarán en menor o mayor medida, o no aflorarán, dependiendo de la personalidad del individuo que a su vez impacta en éstas de dos formas: la primera es en el modo (aumentar, disminuir o no cambiar) y la segunda es el grado o fuerza (poco, mucho, etc.). Los estados físicos también pueden influir en las emociones del individuo, de modo que estados como el hambre, por ejemplo, puede disminuir el grado de alegría. Finalmente, las actitudes, también se ven influenciadas por la personalidad y a la vez pueden modificar de modo considerable las emociones.

Para hacer que el comportamiento del agente sea razonable y creíble, se debe "recordar" de alguna forma lo que ha ocurrido en el pasado en torno al agente, para ello se cuenta con dos mecanismos que son: el mantenimiento acumulativo del estado pasado y el mantenimiento explícito del estado pasado o historia pasada que debe incluir la proposición relativa al suceso acaecido, el instante temporal y la importancia del suceso.

Una vez conocidas las creencias y las influencias entre ellas, cualquier percepción recibida por los sensores puede ser incorporada a las creencias del agente sobre algo en particular y podrá ser el disparador de alguno de los procesos de razonamiento, ser agregada a la historia pasada del agente o alguna combinación de éstos.

Otro aspecto importante a considerar en este nivel reactivo de COGNITIVA, son los intereses del agente y cómo mantenerlos. Para ello, se plantea introducir un elemento que expresará explícitamente los límites entre los que se desea que se encuentren los estados transitorios del individuo, de modo que, se pueda obtener los comportamientos oportunos sin alterar la inmediatez de las respuestas reactivas. Como es de esperar, los rasgos de personalidad también ejercen influencia en los intereses haciendo que los límites sean menores o mayores, es decir los deseos de comportamiento propuestos por otros niveles de la arquitectura se manifestarán en los intereses del agente en la medida en que sus rasgos de personalidad se lo permitan. Finalmente, el procesamiento reactivo de los preceptos puede realizarse de dos formas que son (Imbert, 2005):

1. Procesado de reflejos: A partir de las creencias e intereses actuales del agente, éste es capaz de producir respuestas adecuadas con una carga mínima de voluntariedad. En función de los parámetros disparadores, justificadores y acciones respuesta.

2. Procesado de reacciones conscientes: Debe ser capaz de reaccionar ante situaciones originadas, bien por la ocurrencia de eventos o la llegada de información, o bien por las creencias que se tiene acerca del estado actual, siempre acorde con sus intereses específicos. Este tipo de comportamiento implica un grado ligeramente mayor de consciencia en la reacción, y puede suponer la ejecución de una o varias acciones, a veces con forma de mini-planes reactivos preconcebidos.

Nivel deliberativo

El nivel deliberativo se incluye para dotar al agente de la capacidad de generar planes de comportamientos con una visión temporal más amplia y explotando su capacidad autónoma. En COGNITIVA el proceso deliberativo se sustenta sobre dos pilares fundamentales (Imbert, 2005). El primero de ellos son las *metas* que tiene el agente, que manifiestan los objetivos a medio-largo plazo y orientan las acciones en el futuro. Es importante distinguirlas de los intereses que, en vez de objetivos, especifican el rango dentro del cual se desea que se encuentre cada estado transitorio del agente. En COGNITIVA una meta está compuesta por: una *situación objetivo* que es el estado que el agente aspira alcanzar un *estado actual* que representa cómo se encuentra una meta en un momento (pueden ser: en espera (E), en procesamiento (R), inalcanzable (I), planificada (P), en ejecución (J), cancelada (C) o alcanzada (A)), su *importancia* o trascendencia, el *tiempo de generación,* que es cuando la meta adquiere el estado "en espera" y comienza el ciclo de vida y, finalmente, su *caducidad,* que es el tiempo máximo para alcanzar la meta. El conjunto de metas en un instante determinado se denomina *Goal* (Imbert, 2005).

El encargado de generar y gestionar las metas se denomina *generador de metas deliberativo,* que tiene en cuenta diversos criterios para organizar las metas existentes con respecto a las nuevas, e incluso, cancelando las que considere obsoletas.

El segundo pilar del nivel deliberativo corresponde a los planes compuestos por *metas*, *acciones* y una *valoración* que viene dada por parámetros particulares de cada contexto que serán significantes para el organizador cuando ha de optar por uno de los planes propuestos para la misma meta. Los planes son elaborados por el *planificador deliberativo*, que realiza lo siguiente: a partir de una meta en estado "en espera", irá proponiendo acciones para alcanzarla, de acuerdo con el algoritmo de planificación elegido. A su vez, el planificador del nivel social colaborará también en la elaboración de dicho plan, proponiendo acciones apropiadas desde su perspectiva, que se intercalarán con las propuestas por el planificador deliberativo. El *planificador deliberativo* tiene a su cargo también la actualización de los intereses para mantener la coherencia de los comportamientos reactivos.

Nivel social

El nivel social surge, como es lógico, de la interacción de otros y de la necesidad de que esto se manifieste en las acciones planificadas por el agente. Las metas, desde el punto de vista social, serán propuestas por el agente para satisfacer necesidades e intereses relativos a su interacción con otros individuos (Imbert, 2005).

Del mismo modo que para el nivel deliberativo, este nivel cuenta con el *generador de metas social,* cuyas metas se diferencian de las deliberativas en su alcance y serán gestionadas por el *planificador social,* que al contrario de competir con el *planificador deliberativo*, deberán cooperar para proponer las acciones que creen que son las más oportunas.

En la Figura 2 se muestra la disposición interna de la arquitectura completa de COGNITIVA.

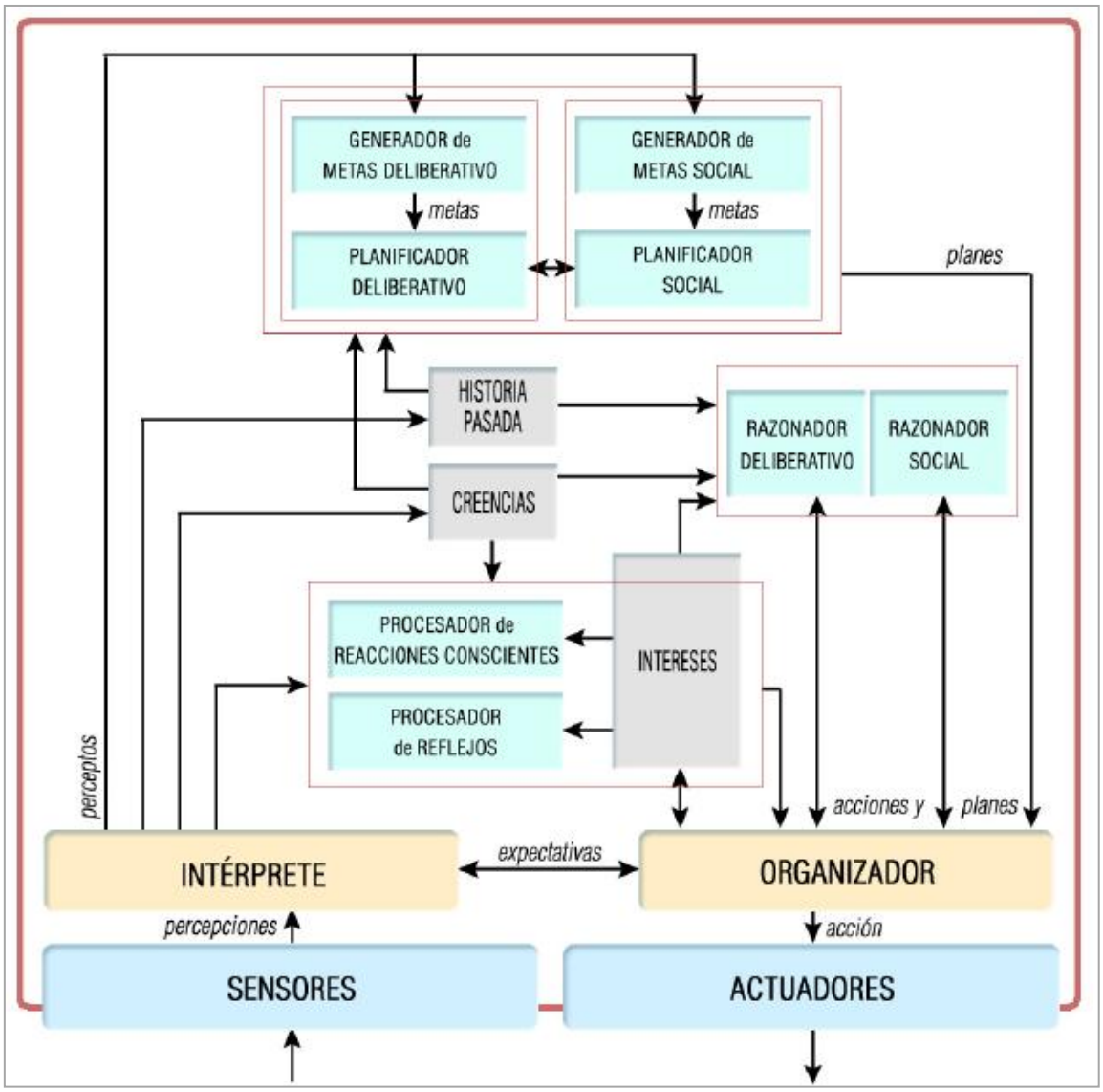

Figura 2 – Arquitectura completa de COGNITIVA.

Se ha mencionado que COGNITIVA supera los problemas de arquitecturas demasiado específicas o muy generalistas y, para ello, plantea desarrollar dos tipos de concreción al momento de abordar un problema. Éstas se detallan a continuación.

Concreción funcional

La concreción funcional está orientada a definir todas las funciones "primitivas" necesarias para el funcionamiento de COGNITIVA sin considerar el contexto del problema.

Según el autor de COGNITIVA, es posible realizar un sin número de concreciones funcionales que tendrán cierta *homogeneidad,* ya que deben especificar como mínimo el *dominio y aritmética de operación*, las *creencias* –básicamente las que pertenecen al modelo personal del agente–*,* la *historia pasada*, los *intereses,* las *acciones y el organizador*, las *expectativas*, el *intérprete de percepciones*, las *metas y planes*, el *comportamiento reactivo*, el *comportamiento deliberativo* y por último el *comportamiento social*.

El objetivo de la concreción funcional es asignar el tipo de valores de los elementos de la arquitectura y su caracterización o estructuración. La Tabla 2 muestra algunos ejemplos de aplicar la concreción funcional en algunos de estos elementos.

Elemento	Concreción funcional
Creencias	*concepto-atributo-valor* (A, *sed*, <bastante>)
Relaciones entre creencias del modelo personal	influyente MODO_de_INFLUENCIA <grado_de_influencia> influido (optimismo AUMENTA <mucho> alegría)

Tabla 2 – Ejemplos de la Concreción Funcional.

Concreción contextual

La concreción contextual, como su propio nombre lo dice, consiste en particularizar la concreción funcional desarrollada de acuerdo con el contexto real del problema. Dividir la concreción funcional de la contextual es lo que hace de COGNITIVA una arquitectura potente para el diseño de aplicaciones de cualquier contexto sin tener que volver a cero cada vez que se tenga un dominio diferente.

Los elementos que deberán contextualizarse son los siguientes:

- Creencias, que serán las que se emplearán según el dominio, sus relaciones y el valor de reposo de cada una.
- Historia pasada, que determina los acontecimientos que serán registrados.
- Intereses, precisando los umbrales superior e inferior para mantener ciertas creencias.
- Acciones que podrá ejecutar el agente.
- Expectativas, que pueden ser procedentes de una acción o introducidas directamente por el diseñador, además de especificar la variación que se producirá por las expectativas.
- Perceptos que gestionará el intérprete.
- Comportamiento reactivo, a modo de comportamientos reflejo como reacciones conscientes.
- Metas predefinidas y metas generadas dinámicamente.
- Comportamiento deliberativo o mecanismos para la generación de metas.
- Comportamiento social o mecanismos para la generación de metas teniendo en cuenta la interacción con otros.

2.2 Algoritmos Genéticos

La otra tecnología en la que se va a sustentar la solución propuesta en el presente trabajo de tesis de máster, además de la de los agentes, es la de los algoritmos genéticos.

Los Algoritmos Genéticos se enmarcan dentro de la Computación Evolutiva, que es una rama de la Inteligencia Artificial y que, como su propio nombre indica, se basa en la evolución biológica para desarrollar técnicas que puedan dar solución a problemas complejos que otros métodos no serían capaces de solucionar en un tiempo razonable. En conjunto con la Programación Evolutiva, las Estrategias Evolutivas y la Programación Genética, los Algoritmos Genéticos forman los cuatro paradigmas fundamentales de los Algoritmos Evolutivos, que engloban a todas las técnicas o métodos que tienen como base a los mecanismos evolutivos. Las principales diferencias entre estas técnicas se muestran en la Tabla 3.

Algoritmo	Representación del problema	Operadores de variación	Métodos de selección
Algoritmos genéticos (Goldberg)	Cadena binaria	Mutación y crossover	Selección de rueda de ruleta (a veces con elitismo)
Estrategia de Evolución (Rechenberg / Schwefel)	Vector de reales + desviaciones estándar	Mutación Gaussiana y crossover aritmético (diferentes tipos)	Diferentes tipos de selección: λ, μ, λ + μ
Programación Evolutiva (Fogel)	Número reales	Mutación	Diversos tipos
Programación genética (Koza)	Expresiones-S de LISP representadas habitualmente como árboles	Crossover, algo de mutación	Diversos tipos

Tabla 3 – Diferencias entre Algoritmos Evolutivos

Como ya se habrá podido notar, todas estas técnicas hacen uso de los términos relacionados con la evolución biológica, conservando en cada caso los mismos objetivos y significado; pero, obviamente, con variaciones propias de la implementación en un entorno computacional. Por ejemplo, al igual que el término *crossover* se usa en biología para expresar la reproducción sexual de dos individuos diferentes en los que los hijos se quedan con parte del material genético de cada progenitor, en la computación evolutiva también se usa para generar nuevas configuraciones, ya sean cadenas binarias, árboles u otros, a partir de otras dos que son consideradas los padres.

Tal como lo mencionan algunos autores (Goldberg, 1989; Holland, 1968), los algoritmos genéticos difieren de los métodos tradicionales en varias formas:

- Utilizan funciones de objetivo para determinar la aptitud del individuo (que es un punto del dominio) a evaluar, no derivadas de otro tipo de información adicional. Los Algoritmos Genéticos son ciegos.
- No trabajan con variables de diseño, sino con una codificación de éstas.

- Utilizan un conjunto de puntos del dominio en contraposición con los métodos tradicionales, que se basan en un único punto. Esto se traduce en que en cada iteración los Algoritmos Genéticos procesan y evalúan un determinado número de diseños.
- Utilizan reglas u operadores estocásticos en lugar de las tradicionales reglas determinísticas.
- Existe la certeza matemática de que el algoritmo es capaz de obtener siempre un óptimo global, si no existe una limitación temporal en el cálculo.

Estos algoritmos tienen como principales ventajas el ser robustos y equilibrar eficiencia y eficacia en la determinación de puntos óptimos, lo cual les permite ser ideales para entornos altamente cambiantes o restringidos.

Cualquier algoritmo genético estará compuesto principalmente de cinco componentes (Michalewicz, 1992), que son: una representación de soluciones potenciales del problema, un método para crear la población inicial, una función de evaluación que establece un orden de las soluciones en términos de adecuación, operadores genéticos que alteran la composición de los hijos y valores de los parámetros de configuración del propio algoritmo (tamaño de la población, probabilidades de aplicar operadores genéticos, etc.)

En los siguientes apartados se describe cada uno de los aspectos más relevantes de los Algoritmos Genéticos, debido a que éstos serán empleados como parte de la solución propuesta de este trabajo.

2.2.1 Origen

La idea en la cual se ha inspirado el desarrollo de los algoritmos genéticos corresponde, nada más que a la misma selección natural enunciada por Charles Darwin en su libro “El Origen de las Especies”, en la que se plantea que los individuos más aptos de una población son los que sobreviven, ya que se adaptan más fácilmente a los cambios de su entorno. El propio Darwin reconoce que el azar en la variación, junto con la ley de la selección, es una técnica de resolución de problemas de inmenso poder y de aplicación casi ilimitada.

Esto dio origen a que, a finales de los 50’s y principios de los 60’s, biólogos evolutivos que buscaban realizar modelos explícitos de la evolución natural crearan ejemplos, aunque sin mucha trascendencia, de lo que serían hoy algoritmos genéticos. En esta misma época, otras investigaciones que no generaron gran reacción fueron las de G.E.P. Box, G.J. Friedman, W.W. Bledsoe y H.J. Bremermann, que en 1962, desarrollaron de forma independiente algoritmos inspirados en la evolución para optimización de funciones y aprendizaje automático.

Mayor éxito tuvo la el trabajo de Ingo Rechenberg, quien en 1995 introdujo una técnica a la cual denominó “estrategia evolutiva”. En ésta, un padre mutaba para producir un descendiente, y se conservaba el mejor de los dos, convirtiéndose en el padre de la siguiente iteración. En las primeras versiones esta técnica no hacía uso de los conceptos de población ni cruzamiento incluyéndose el primero de éstos en versiones posteriores.

Un importante avance se produjo en 1966, año en el que L.J. Fogel, A.J. Owens y M.J. Walsh desarrollaron la técnica denominada “programación evolutiva” en la que se introduce por primera vez el concepto de población, lo que permitía que la salida de resultados no

dependiera sólo de la entrada de datos actuales sino también de las anteriores. Según esta técnica, las soluciones candidatas para los problemas se representaban como máquinas de estado finito sencillas; al igual que en la técnica anterior, el algoritmo funcionaba mutando aleatoriamente una de estas máquinas simuladas y conservando la mejor de las dos (Mitchell, 1996; Goldberg, 1989).

Ya desde 1962, John Holland venía interviniendo en esta área de investigación publicando trabajos relacionados con sistemas adaptativos. Es hacia finales de los 60's que desarrolló una técnica que imitaba en su funcionamiento a la selección natural, que fue denominada originariamente "planes reproductivos", y que con la publicación de su libro "Adaptación en Sistemas Naturales y Artificiales" en 1975, se comienzan a denominar Algoritmos Genéticos. Éstos ya implementan procesos de mutación, selección y cruzamiento, simulando el proceso de la evolución biológica como estrategia para resolver problemas.

Sin dejar pasar mucho más tiempo, entre principios y mediados de los 80's, los algoritmos genéticos se estaban aplicando ya en una amplia variedad de áreas, desde problemas matemáticos abstractos hasta asuntos tangibles de ingeniería como el control de flujo en una línea de ensamble, reconocimiento y clasificación de patrones y optimización estructural (Goldberg, 1989), y hacia 1998, éstos ya habían dado el salto del mundo científico al sector comercial resolviendo problemas de interés cotidiano (Haupt y Haupt, 1998).

2.2.2 Definición

Son varios los autores que han publicado trabajos relacionados con los principios básicos de los Algoritmos Genéticos de Holland, ofreciendo diferentes definiciones y perspectivas que se detallan a continuación.

Uno de los personajes más representativos de la comunidad científica avocada a la investigación de los Algoritmos Genéticos es David Goldberg, quien ha dedicado un libro a describir cómo y porqué los algoritmos genéticos funcionan. La definición que él plantea describe de forma muy apropiada el qué –algoritmos de búsqueda– y el cómo –basados en la mecánica de la selección natural y de la genética natural– de los algoritmos genéticos.

> *"Los Algoritmos Genéticos son algoritmos de búsqueda basados en la mecánica de la selección natural y de la genética natural. Combinan la supervivencia del más apto entre estructuras de secuencias con intercambio de información estructurado, aunque aleatorizado, para constituir así un algoritmo de búsqueda que tenga algo de las genialidades de las búsquedas humanas." (Goldberg, 1989)*

Por su parte Z. Michalewicz, coincide con Goldberg en que los algoritmos genéticos son una clase métodos de búsqueda; sin embargo es cuestionable que éstos sean del todo independientes del dominio, ya que existen parámetros y configuraciones imprescindibles para la aplicación de algoritmos genéticos que dependen enormemente del dominio del problema.

> *"Los algoritmos genéticos son una clase de métodos de búsqueda de propósito general (independientes del dominio) que atacan un notable balance entre exploración y explotación del espacio de búsqueda." (Michalewicz, 1992)*

Aunque se relacione más a John Koza con la Programación Genética, aporta también una definición más completa que las anteriores, ya que además de dar a conocer la filosofía básica de los Algoritmos Genéticos, incluye términos más relevantes e ilustrativos como son *operaciones genéticas*, *cadenas de cromosomas* y *función matemática que refleja su aptitud.*

> *"Es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas, de entre las que destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas, y se les asocia con una cierta función matemática que refleja su aptitud." (John Koza)*

Como se mencionó anteriormente, los Algoritmos Genéticos no tardaron mucho en generar interés fuera de la comunidad científica, cuestionando las percepciones clásicas acerca de los mismos y promoviendo el uso de otros componentes que mejoren y/o faciliten la obtención de resultados. Así lo manifiesta Lawrence Davis en su libro "Handbook of Genetic Algorithms", que se enmarca en el campo de la ingeniería, cuyo principal objetivo es aplicar Algoritmos Genéticos a problemas del mundo real y no a aprender sobre sistemas naturales. La filosofía de Davis es moverse lejos del realismo biológico. Por ejemplo, se recomienda el uso de representaciones numéricas y métodos de búsqueda híbridos, y aboga por el mantenimiento de estadísticas explícitas en el éxito de cruce y mutación para variar adaptativamente sus ratios.

A efectos de esta investigación, se considera que la definición de Koza es la que más se asemeja a lo que se describirá en los siguientes apartados.

2.2.3 Terminología de los Algoritmos Genéticos

Los algoritmos genéticos usan un vocabulario "prestado" de la genética natural (Michalewicz, 1992). Por ejemplo, se habla acerca de individuos (genotipos o estructuras) en una población, que con frecuencia se denominan cadenas o cromosomas. Los cromosomas son vectores de *genes* cuyos valores asignados se denomina *alelos*. Un genotipo sometido al medio ambiente se denomina *fenotipo,* y representa un punto en el espacio de soluciones al problema. Los *genes* (también llamados caracteres o decodificadores) se hallan dispuestos en sucesión lineal en la que cada gen controla la herencia de una o más caracteres. Estos términos se describen en los siguientes apartados.

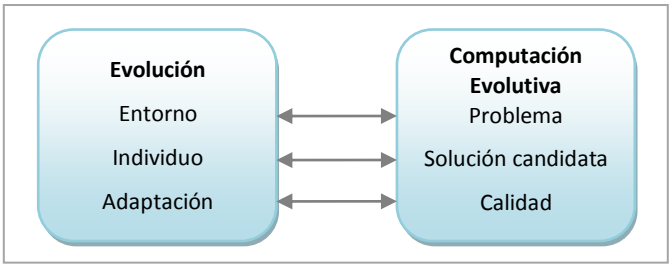

Figura 3 – Metáfora Evolutiva

2.2.3.1 La población

Se puede asumir que el concepto de población es bastante conocido en el mundo real; el punto común de las definiciones del diccionario de la Real Academia Española mencionan, tanto explícita como implícitamente, que una población es un conjunto de individuos. Acorde con lo que se ha mencionado anteriormente acerca de la terminología, en el mundo de los algoritmos genéticos, la población también se refiere a un conjunto de individuos, pero éstos representarán soluciones del problema a resolver.

Atendiendo al proceso propio de los algoritmos genéticos basado en iteraciones, la población no es estática, lo que hace que a lo largo de este proceso se vayan generando diferentes poblaciones, una para cada iteración. La que corresponde a la primera de ellas se denomina *población inicial* y tiene mucha relevancia en el diseño de un algoritmo genético, ya que dependerá de ésta que se puedan evaluar y, por consiguiente, encontrar las mejores soluciones al problema. Para obtener la *población inicial*, generalmente se seleccionan aleatoriamente a individuos de todo el espacio de búsqueda, a menos que se conozca la proximidad del óptimo local buscando solamente individuos cercanos a éste.

La configuración que la población requiere es muy dependiente del dominio y es una tarea crítica para el algoritmo genético. Consiste en establecer el número de individuos –o tamaño– que conformarán la población, que puede estar relacionado con la complejidad del problema, de modo que, cuanto mayor es el espacio de búsqueda, mayor debería ser el tamaño de la población pero, que a la vez, se debe conjugar con el tiempo de procesamiento y el uso de los recursos computacionales, de modo que se tenga aplicaciones que brinden soluciones aceptables en un tiempo razonable.

A partir de la segunda iteración, las poblaciones se irán generando teniendo como base los individuos de la anterior a los que se aplicarán diversos operadores genéticos, que serán descritos posteriormente, para obtener nuevos individuos.

2.2.3.2 Los individuos

Cuando se habla de algoritmos genéticos, los individuos no son más que las soluciones al problema que se quiere resolver. El término *soluciones* en realidad se refiere, coloquialmente hablando, a las opciones de solución que se tiene para el problema en un momento determinado. Estas opciones serán evaluadas en cada iteración para, finalmente, quedarse con la que mejor resultado tenga según la función de adecuación y que pasará a ser la *solución* final de todo el proceso de aplicación de los algoritmos genéticos.

Existen dos términos relacionados con los individuos muy conocidos en Biología. El primero de ellos es el vocablo *genotipo,* que se define en este contexto, como el conjunto de los genes de un individuo, incluida su composición *alélica*. Este mismo concepto es el que emplean los algoritmos genéticos sobre el *genotipo;* pero acotando aún más su estructura a una representación binaria, en la que cada bit vendría a ser un gen y el valor de cada bit sería un alelo. De este modo, el *genotipo* viene a ser la representación interna del individuo sobre la cual trabajan directamente los operadores genéticos.

El segundo de estos términos es el que se conoce como *fenotipo,* cuyo significado según el diccionario de la Real Academia Española hace mención a que es una manifestación visible del *genotipo* en un determinado ambiente. Desde el punto de vista biológico, hay quienes consideran que no puede emplearse el concepto de "manifestación visible" ya que muchas veces se estudian componentes que no son visibles, cambiando así la definición de *fenotipo* a cualquier característica detectable de un organismo determinado por una intersección entre su *genotipo* y su medio. A efectos de ser usado por los algoritmos genéticos, se dirá que el *fenotipo* es la solución en sí misma. La solución fenotípica representa el modelo o la estructura en que la solución genotípica se decodifica.

Esta decodificación está a cargo de la *función de morfogénesis,* de la cual se debe cuidar que una solución o *fenotipo* no sea representada por varios *genotipos,* ya que se tendría una función degenerada que puede ser perjudicial para el funcionamiento del algoritmo, conduciendo a una convergencia prematura y estancamiento en la población de *fenotipos* mientras la población de *genotipos* es muy diversa.

2.2.3.3 Los genes

Del mismo modo que para la Biología, los genes en programación genética, constituyen la unidad funcional para la transmisión de los caracteres hereditarios, ya que codifica el valor de un solo parámetro para el que se ha proporcionado previamente un conjunto de valores posibles. Se debe prestar atención a este número de valores, debido a que se encuentra directamente relacionado con la complejidad del espacio de búsqueda. Es así que si este número de valores es muy grande, más complejo será el espacio de búsqueda requerido.

Debido a que los genes tienen valores habitualmente binarios, es necesario traducir el conjunto de valores especificado a cadenas de ceros y unos; para ello se cuenta con la *función de codificación o mapeo* que es la encargada de mapear todo punto –o solución– del espacio de soluciones en un genotipo definiendo la estructura que tendrán los genes para representar a los valores reales del problema.

2.2.3.4 La función de aptitud

Se ha mencionado ya en anteriores apartados, que los individuos o soluciones de una población deberán ser evaluados para determinar si son excluidos o se mantendrán en la población de la siguiente iteración. Esto no podría ser de otro modo, ya que la filosofía de los algoritmos genéticos es la supervivencia del más fuerte o, en este caso, el que mejor se adapte según la función de aptitud o también denominada función de *fitness*.

La función de aptitud está definida en el espacio *fenotípico,* por lo que se debe aplicar antes, la *función de decodificación* de los genes, que es la opuesta a la que se mencionó en el apartado anterior. Esto se debe a que tiene el mismo tipo que la función objetivo del problema, que se realiza sobre lo que percibe del *fenotipo*.

La función de *fitness* no es trivial en la aplicación de algoritmos genéticos, ya que será la que guíe el mecanismo de exploración, tal como lo hace el entorno para la evolución biológica. Esto supone que su diseño debe tener en cuenta todos los aspectos relevantes del dominio del problema haciéndola muy dependiente de él.

Se ha mencionado en repetidas ocasiones el término “evaluar” para hacer referencia al grado en que un individuo es buen o mal candidato para ser seleccionado como la solución del problema; sin embargo no se debe confundir la *función de evaluación,* que indica qué tan bueno es un individuo de acuerdo a sus parámetros de configuración. Con la *función de aptitud* se toma en cuenta al resto de la población para determinar que tan bueno es un individuo formando una lista en modo de ranking.

2.2.4 Operadores Genéticos

Los operadores genéticos rigen el comportamiento de los algoritmos genéticos, ya que determinan cómo se va implementando cada uno de los procesos relacionados a estos algoritmos. Los operadores más comunes que se emplean son los de inicialización, selección, cruce y mutación, sin dejar de lado los ajustes necesarios para los diferentes parámetros que serán empleados a lo largo de todo el proceso. Los siguientes apartados están dedicados a dar una visión general de los operadores genéticos más conocidos en cada una de las fases de la aplicación de los algoritmos genéticos. Estas fases se pueden apreciar en la Figura 4.

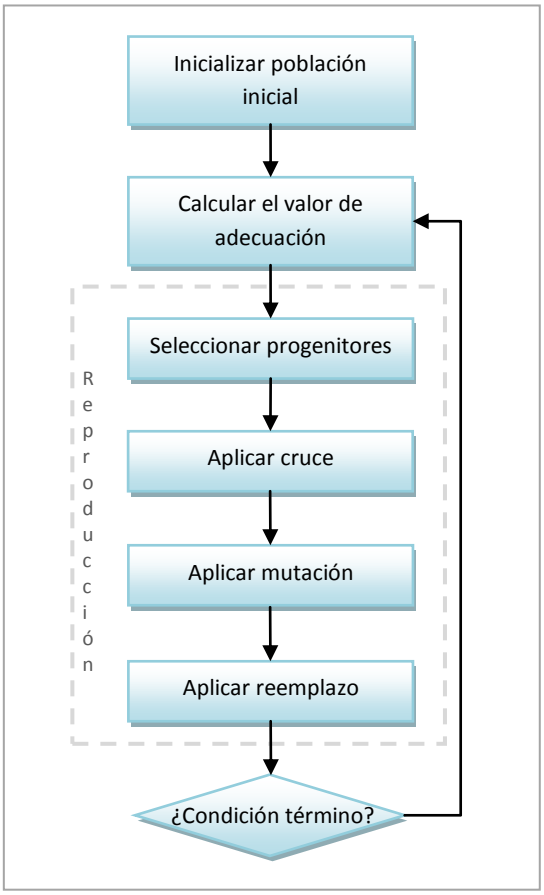

Figura 4 – Proceso general de Algoritmos Genéticos

2.2.4.1 El operador de inicialización

El operador de inicialización está estrechamente relacionado con la población, ya que su objetivo es establecer la primera de ellas o en ocasiones, dependiendo del diseño del algoritmo, reiniciarla.

A muy grandes rasgos, el funcionamiento genérico de este operador consiste en seleccionar, de forma aleatoria, el conjunto de genes que componen a cada uno de los individuos dentro del dominio definido para cada gen. La generación aleatoria no es la única técnica usada para este fin, puede utilizarse otras técnicas heurísticas adaptadas al problema que se desea solucionar, pero se deberá considerar el esfuerzo computacional que ello implica. Además, debe tenerse en cuenta que es conveniente que la población inicial este suficientemente dispersa con el fin de abarcar todo el espacio de búsqueda. Otra opción es introducir soluciones conocidas –sembrar *semillas* buenas– pero se debe cuidar que esto no conduzca a óptimos locales.

2.2.4.2 Los operadores de reproducción

El segundo proceso en la aplicación de los algoritmos genéticos es el de la reproducción, cuyo objetivo es el de generar nuevos individuos a partir de la población actual empleando los diversos operadores como la selección, el cruce, la mutación o el reemplazo. La finalidad de aplicar estos operadores es el de generar mejores individuos, que se vayan adaptando al problema y a su vez puedan dar lugar a nuevas generaciones –hijos– aún mejores que ellos mismos. Esto se conoce como *explotación* y se define como el proceso de utilizar la información de puntos del espacio de búsqueda previamente visitados, para determinar los puntos que conviene visitar a continuación. No se debe confundir este proceso con el de *exploración* que está relacionado con la visita de nuevas regiones del espacio de búsqueda y la selección de nuevos puntos que puedan ser soluciones prometedoras. Estos procesos deben ser ajustados teniendo en cuenta el tipo de problema a resolver y sabiendo que una *exploración* demasiado intensiva hace el proceso de búsqueda más lento debido a que no se generan suficientes esquemas; mientras que una *explotación* excesiva puede provocar que se converja en un óptimo relativo.

La fase de reproducción presenta tres pasos y para cada uno de ellos se cuenta con su operador respectivo. Estos operadores se describen a continuación.

El operador de selección

Como es lógico, el primero de los tres pasos de la reproducción es la selección de los padres que van a reproducirse que, como es de esperar, es uno de los procesos más críticos de los algoritmos genéticos, ya que de ellos depende que se llegue a encontrar soluciones cada vez más acertadas al problema que se desea resolver.

La selección tiene bajo su responsabilidad elegir a los individuos que tendrán más oportunidades de reproducirse y descartar a los que no, otorgando mayor peso a aquéllos que son más aptos según su valor de ajuste. Esto se corresponde perfectamente con el proceso *inspirador* –la selección natural. El hecho de que se busque a los individuos más aptos no significa que los menos aptos deban ser totalmente descartados desde el principio, éstos pueden contener entre sus genes porciones con gran potencialidad aunque que en su totalidad no sea considerado el más apto. Por ejemplo, una opción sería aplicar las técnicas de selección para el primer padre y seleccionar aleatoriamente el segundo. El primero de ellos se selecciona atendiendo a diversos criterios como la selección de los mejores individuos, la eficiencia computacional, entre otros. Éstos pueden ser agrupados según el modo de selección

que puede ser directo, en el que los padres se eligen de acuerdo a un criterio objetivo, o aleatorio que a su vez deja abierta la opción de una selección *equiprobable*, en la que todos tienen la misma probabilidad de ser escogidos, o *estocástica* que dependen de una heurística determinada. Los tipos de selección se detallan en las siguientes líneas.

La ***selección por ruleta*** introducida por De Jong, es posiblemente el método más utilizado desde los orígenes (Blickle and Thiele, 1996). Otras formas de referirse a ella son *muestreo estocástico con reemplazo* o *selección proporcional de la aptitud,* ya que la probabilidad de selección es directamente proporcional a su aptitud. Su funcionamiento se basa en una ruleta en la que los mejores individuos recibirán una porción mayor que la recibida por los peores, la suma de todos estos porcentajes debería ser la unidad, las porciones más grandes se ubican al inicio de la ruleta. Para seleccionar un individuo basta con generar un número aleatorio entre [0..1] y devolver el individuo situado en esa posición de la ruleta. Los inconvenientes que presenta esta técnica es que se vuelve ineficiente a medida que aumenta el tamaño de la población y que el peor individuo puede ser seleccionado más de una vez.

La ***selección por muestro universal estocástico*** o *selección estocástica sin remplazo* definida originalmente por De Jong y estudiada por Baker. Fue creada para mejorar la mala distribución de los individuos en función de los valores esperados obtenidos según el operador anterior. Su funcionamiento consiste en mapear los individuos en segmentos contiguos de una misma línea, de manera que cada segmento es proporcional en tamaño a su aptitud y colocar un número de punteros, igual al número de individuos a seleccionar, de forma equidistante. Se van seleccionando los individuos que coinciden con la posición indicada por un número aleatorio comprendido entre 0 y el número de porciones. Sus ventajas son básicamente que reduce la variabilidad estocástica, reduce el tiempo de computación y garantiza que un individuo apto puede ser seleccionado incluso varias veces.

La ***selección por muestreo determinístico*** es una variación del anterior y obtiene también el número de veces que se espera que un individuo sea seleccionado para la reproducción. La parte entera de este valor se usa para determinar el número de veces que se seleccionará cada individuo y la parte decimal indica el orden de la población. Los individuos restantes se determinan en función del ordenamiento anterior.

Una variación de la *selección por muestreo determinístico* es la ***selección por muestreo estocástico del resto con remplazo,*** en la que se seleccionan en función de la parte entera y la parte no entera se usa para calcular la probabilidad de selección según el procedimiento de la ruleta.

Otra variación del *muestreo determinístico* es la ***selección por muestro estocástico del resto sin reemplazo,*** en la que se usa la parte entera para seleccionar los individuos –igual que el anterior– pero la parte no entera se emplea para lanzar monedas donde la probabilidad de que se obtenga cara o sello está ponderada por el valor no entero. Un ejemplo de ello es que si el valor esperado es 1,8 significa que este individuo será seleccionado al menos una vez y tendrá una probabilidad del 80% de ser seleccionado. Booker ha demostrado la superioridad de esta técnica frente al resto de operadores estocásticos.

La ***selección por grupos*** o por *bloques* fue propuesta por Thierens y Goldberg y se implementa dividiendo la población en un cierto número de grupos y asignándoles una probabilidad de selección, que será dividida por el número de individuos para obtener la probabilidad de selección de un individuo.

La ***selección por rango*** fue definida por Baker para superar los inconvenientes de la *selección por ruleta*. La probabilidad de selección de un individuo se determina en función del rango (en lugar de la aptitud) que éste posee dentro de la población. Este rango se determina asignando el valor 1 al individuo con peor aptitud y un valor mayor que 1, determinado en función del tamaño de la población, a los individuos más aptos. La selección se realiza de forma aleatoria mediante un torneo relacionado con el rango que determinará el vencedor.

La ***selección por torneo*** por su parte fue propuesta por Wetzel y se hizo popular por Deb y Goldberg, ya que es una de las más efectivas y sencillas de implementar. Consiste en comparaciones directas entre los individuos, seleccionado dos o más y haciendo que compitan entre sí, venciendo el individuo con mayor aptitud. Se han presentado dos versiones de esta técnica. La primera de ellas es la versión *determinística,* que selecciona al azar un número de individuos (generalmente 2) para luego escoger al más apto de éstos; la segunda es la versión *probabilística* que no selecciona la mejor según su aptitud, sino que obtiene un número aleatorio entre [0..1] y que si éste es mayor que un parámetro de configuración general de todo el proceso (este valor es usualmente mayor a 0,5 y menor o igual que 1), se escoge al individuo más apto, de lo contrario se seleccionará al que tenga menor aptitud.

El operador de cruce

En el paso anterior se han obtenido los padres que se deberán reproducir para generar nuevos individuos, que persiguen ser mejores que sus progenitores. Para ello, se emplea el operador de cruce, también denominado recombinación o *crossover*, que en líneas generales consiste en el apareamiento de dos individuos con ciertas características distintas y deseables para obtener una descendencia que comparta genes de ambos padres, ya que existe la posibilidad de que la combinación de estos genes sean precisamente los causantes de la bondad de los padres. Al compartir las características buenas de dos individuos, es acertado pensar que la descendencia podría tener una bondad mayor que cada padre. Además, mientras la diferencia de los padres sea mayor, será mayor la influencia de este operador en el funcionamiento del algoritmo.

Las operaciones que realiza este operador son básicamente: escoger dos individuos previamente seleccionados por el operador de selección, generar un número aleatorio entre 0 y 1 y comparar este valor con la probabilidad de cruce. Si este número es menor, se seleccionan uno o varios puntos de las cadenas de los padres y generan dos hijos. Si el número obtenido es mayor a esta probabilidad, simplemente se copian los padres.

El *punto de corte* se refiere a la posición a partir de la cual se tomarán las características del primer padre para unirse a las características del segundo padre posteriores a esta posición (ver Figura 5). Éste puede realizarse por genotipo –cortando cualquier punto de la cadena de genes– o por fenotipo, realizando el corte entre bloques de genes. La determinación del *punto de cruce* genera los operadores que se describen en las siguientes líneas.

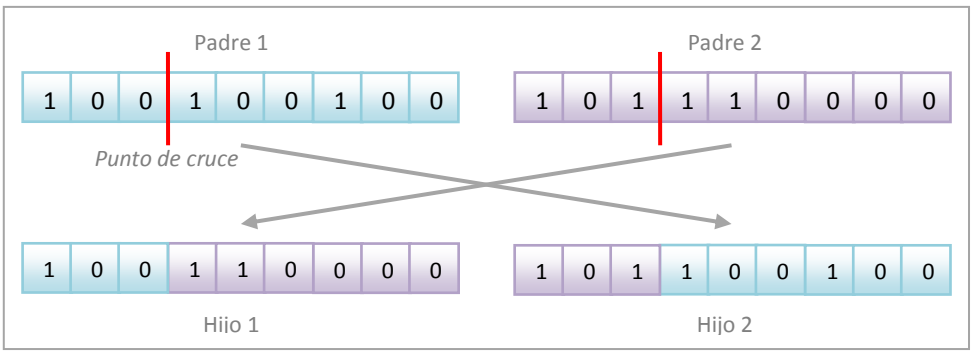

Figura 5 – Operador de cruce

Los operadores de cruce *determinísticos* son aquellos en donde los genomas de los hijos se obtienen mezclando los genes de los padres. Dentro de éstos se encuentra el ***cruce por un punto,*** que es el operador más simple. Fue propuesto por Holland y estudiado por De Jong y consiste en que se cortan los cromosomas por un punto, que es seleccionado aleatoriamente, para generar dos segmentos en los que se diferencia la *cabeza* y la *cola*, para intercambiar las *colas* y generar los nuevos descendientes. Esta técnica puede generalizarse fácilmente a *n* puntos y se denomina ***cruce por n puntos,*** que en vez de cortar por un único punto, se realizan *n* cortes teniendo en cuenta que ninguno de estos puntos coincida con el extremo. Esto genera la ventaja de combinar un mayor número de esquemas; pero estudios desaprueban esta técnica (Jong, 1975) ya que a mayor número de puntos de cruce se reduce el rendimiento del algoritmo.

Otros operadores *determinísticos* son el ***cruce uniforme*** desarrollado por Syswerda, que emplea una máscara de cruce con valores binarios donde si se encuentra el valor 1, el gen situado en esta posición se copia del primer padre y si hay un 0 se copia del segundo; de este modo todos los genes tienen igual probabilidad de ser seleccionados. Además se encuentra el ***cruce plano*** definido por Radcliffe para minimizar las disrupciones producidas por el *cruce uniforme.* Está basado en el concepto de *schemata* (esquemas) binarios planteados por Holland, que es un patrón común entre dos o más individuos de una población.

Otro grupo de operadores de cruce son los *aritméticos* que son más apropiados en codificaciones de punto flotante y se fundamenta en una analogía con la teoría de conjuntos convexos. Dependiendo del número de genes combinados se definen tres tipos de recombinación aritmética. El primero de ellos es la ***recombinación simple,*** que selecciona un gen aleatorio del genoma y se copia el genoma del primer padre hasta la posición del gen obtenido, completando el resto, por medio de algún operador anterior. En la ***recombinación aritmética completa***, todos los genes de los hijos se forman mediante la combinación de ambos padres.

Existen más operadores de cruce como por ejemplo el de ***cruce geométrico*** definido por Michalewicz, que genera un hijo por cada pareja cuyos genes se forman mediante la combinación cuadrática de ambos padres. El ***cruce por mezcla alfa*** propuesto por Eshelman y Shaffer, que utiliza la distancia rectilínea o de Manhattan para generar los genes de los hijos. Finalmente, el ***cruce binario simulado*** definido por Deb, para simular el comportamiento del

operador de cruce por un punto en codificaciones binarias y de diseñar un operador para codificaciones reales.

Hasta ahora se han visto operadores de cruce por un solo punto, pero pueden aplicarse también operadores *multipunto*. Dentro de este grupo se encuentra el ***cruce unimodal de distribución normal*** que fue propuesto por Ono y Kobayashi para una combinación de 3 padres y fue ampliado por Kita a *m* padres. Otro de ellos es el ***cruce parento-céntrico,*** desarrollado por Deb, que es una variación del anterior y en el que se generan los hijos en el espacio alrededor de los padres. El último de los operadores de cruce es el ***cruce simplex,*** propuesto de forma independiente por Tsutsui y Goldberg y por Renders y Bersini, que genera los hijos sobre la superficie comprendida entre los padres.

El operador de mutación

El operador de mutación es el responsable de expandir el espacio de búsqueda fuera del delimitado por los padres, lo que ha sido definido anteriormente como *exploración,* siendo éste el objetivo de realizar mutaciones y no el perfeccionamiento de los hijos con respecto a los padres. Del mismo modo que las mutaciones naturales, los hijos mutados tienen una aptitud muy inferior a la de los padres. Esto determina que el porcentaje de mutaciones no debe ser muy elevado.

A diferencia de los operadores de cruce, que trabajan sobre dos o más individuos, los de mutación son operadores unarios, ya que sólo actúan sobre un individuo. La forma de selección de los genes a mutar parte de la generación de un número aleatorio entre [0, 1] de modo que, si este valor es menor o igual a la probabilidad establecida se producirá la mutación de ese gen. Ver Figura 6.

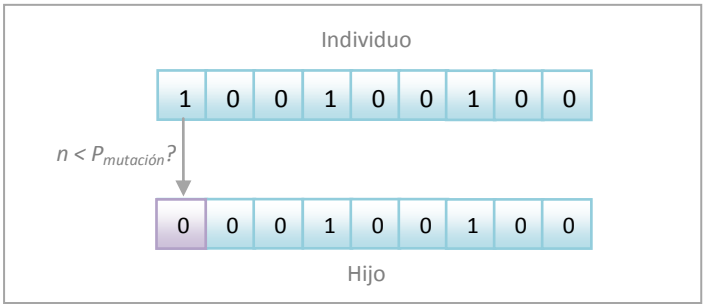

Figura 6 – Operador de mutación

Ejemplos de los operadores de mutación son la ***mutación aleatoria uniforme***, descrita por Michalewicz que, en caso se haya decidido mutar un gen, se generará un número aleatorio dentro del dominio de dicho gen para ser reemplazado por este valor obtenido. Este operador presenta la dificultad de que el valor de reemplazo puede estar muy alejado del gen inicial lo cual no facilita la *explotación* del algoritmo. El operador de ***mutación aleatoria no uniforme*** pretende corregir esto, sacrificando la *exploración* en beneficio de la *explotación.* Este operador establece la proporción de individuos alejados del gen inicial y a continuación, si el gen es elegido para sufrir una mutación se genera un número aleatorio –simulando lanzar una moneda al aire– que si se obtiene cara se suma uno al valor inicial del gen y si sale cruz se resta

uno. Ésta fue propuesta para números enteros y Michalewicz propone una variación aplicable a todos los reales.

Uno de los operadores más empleados en la actualidad, es la ***mutación de convolución gaussiana*** que se realiza mediante la adición de ruido gaussiano al gen que se desea alterar. Este operador sitúa el gen modificado próximo al gen inicial en la mayoría de los casos.

El operador de reemplazo

Llegado este punto, se tiene ya la nueva descendencia conformada por el resultado de aplicar los operadores de selección, cruce y mutación. La tarea de los operadores de reemplazo es determinar qué individuos pasarán a conformar la población de la siguiente iteración. Para ello, se proponen dos clases fundamentales de algoritmos genéticos dependiendo de la estrategia de reemplazo que se vaya a utilizar. Se denominan **Algoritmos Genéticos Simples** o con reemplazo generacional en los que la población de padres es completamente reemplazada por la población de hijos y deben emplear operadores elitistas para evitar la pérdida de los mejores individuos. Por otro lado, los **Algoritmos Genéticos con brecha generacional** sí mantienen una proporción de padres dando lugar a los siguientes operadores de reemplazo.

El ***reemplazo de los menos aptos,*** definido por De Jong y Sarma, en la que los hijos engendrados reemplazan a los individuos menos aptos de la población.

El ***reemplazo aleatorio,*** propuesto por Syswerda, en el que los hijos engendrados reemplazan de forma aleatoria a los individuos de la población.

El ***torneo a muerte,*** definido por Smith, selecciona aleatoriamente a un conjunto de individuos de la población, donde el menos apto es sustituido por un hijo.

El ***reemplazo del individuo más viejo,*** que como su nombre lo dice, se reemplaza al individuo más antiguo, aunque se puede eliminar al más apto si no se incluye algún operador de elitismo.

La ***selección conservativa,*** propuesta por Smith, combina la estrategia de reemplazo del más viejo con un torneo de selección binaria realizando un torneo entre el individuo más viejo con otro seleccionado aleatoriamente.

El ***reemplazo de los progenitores*** se basa en la aptitud, distancia entre genomas u otros métodos para decidir si el hijo reemplaza a alguno de los padres o bien a ambos.

Finalmente el ***elitismo*** evita la eliminación del individuo o conjunto de individuos más aptos de la población. Es recomendable a tal medida que casi todos los operadores de reemplazo y algunos de cruce incorporan estrategias elitistas.

2.2.4.3 Ajuste de los parámetros de los operadores genéticos

Se ha mencionado, al describir los operadores anteriores, que existen diversos parámetros que deben ajustarse para conseguir la eficiencia y eficacia del algoritmo genérico implementado. Este ajuste se puede realizar de forma *offline* u *online*.

El ajuste *offline* tiene un enfoque determinístico y se basa en la experiencia previa en problemas similares. Es muy sencillo de implementar pero no siempre los valores son correctos para el problema. Muchos investigadores se han ocupado de estos ajustes (Holland, De Jong, Greffenstette, Goldberg, Woon entre otros); pero ha sido el trabajo de Hart y Belew y de Wolpert y Macready en el que establecieron el teorema de *no hay café para todos* que determina que, en promedio, para todos los problemas todos los algoritmos de búsqueda, incluida la aleatoria, no funcionan igual. Esto aplicado a los parámetros significa que unos parámetros de ajuste para un problema o conjunto de problemas no funcionarán bien para otros. Los ajustes se realizan fijando todos los parámetros menos uno, que se modifica y genera la curva de eficiencia de ese parámetro; de este mismo modo se continúa con todos los demás. Este enfoque estático no tiene en cuenta las interrelaciones entre parámetros, además es bastante compleja y requiere de un diseño previo de experimentos. Además de ello, el uso de parámetros estáticos conceptualmente está en contra del mismo proceso evolutivo.

Por su parte, en el ajuste *online* es el propio algoritmo el que se encarga de ir ajustando los parámetros. Los mecanismos de control pueden ser de dos tipos. El primero de ellos es la *retroalimentación heurística,* basada en ciertos parámetros que se calculan en cada generación como la convergencia, diversidad, variación de la aptitud de la población, etc. El segundo se denomina *autoadaptación* que incorpora los parámetros en la propia codificación, dejando que sea el propio proceso evolutivo el que se encargue de realizar el ajuste. La Figura 7 recoge los valores de los parámetros principales de los algoritmos genéticos sugeridos por diversos autores.

Autor/es	Población	Probabilidad cruce	Probabilidad de mutación	Número máximo de generaciones
De Jong [83] (1975)	No menciona	0,60-0,80	0,010-0,020	No menciona
Greffenstette [141] (1986)	80	0,45	0,010	20
Wang [407] (1991)	100	1,00	0,010	50
Janson y Frenzel [190] (1993)	100	No menciona	0,001	500
Ball et al. [19] (1993)	100	0,30	0,006	180
Koumousis y Georgiou [214] (1994)	20	0,60	0,010	100
Srinivas y Patnaik [372] (1994)	30-200	0,50-1,00	0,001-0,05	No menciona
Croce et al. [76] (1995)	300	1,00	0,030	3000
Dorndorf y Pesch [98] (1995)	200	0,65	0,001	300
Tate y Smith [385] (1995)	100	0,25	0,750	2000
Keane [203] (1995)	100	0,80	0,005	10
Suresh et al. [379] (1996)	40-60	0,50-0,70	0,100	No menciona
Carroll [58] (1996)	100-200	0,60	0,010-0,005	20
Carroll [58] (1996)	50	0,50	0,020-0,040	12
Roston y R.H. [324] (1996)	200	0,50	0,010	150
Furuta et al. [122] (1996)	No menciona	0,25	0,010	No menciona
Liu et al. [244] (1997)	50	0,80	0,010	100
Roman et al. [323] (2000)	25-125	0,60-0,90	0,005-0,05	100
Nanakorn y Meesomklin [275] (2001)	40	0,8	0,001	100
Nanakorn y Meesomklin [275] (2001)	50	0,85	0,05	100
Annicchiarico y Cerrolaza [12] (2002)	100	1,00	0,020	50
Victoria y Martí [402] (2005)	20	0,8	0,003	300

Figura 7 – Ajuste de parámetros según varios autores

2.2.4.4 Criterios de detención

Acorde con la filosofía de la evolución natural, los algoritmos genéticos podrían ejecutarse indefinidamente si no se establecen criterios que hagan que se detenga y obtengan la solución al problema en un tiempo determinado. Estos criterios están pensados para detectar la convergencia o estancamiento de los individuos, quedando sin sentido que el algoritmo se continúe ejecutando.

Estos criterios deben atender a diversos criterios y sobre todo a la evolución temporal típica de un algoritmo genético, cuya aptitud se ve ampliamente mejorada sólo en las primeras *k* generaciones que luego mejorará muy lentamente implicando el consumo de muchos más recursos computacionales. Esto se ilustra en la Figura 8.

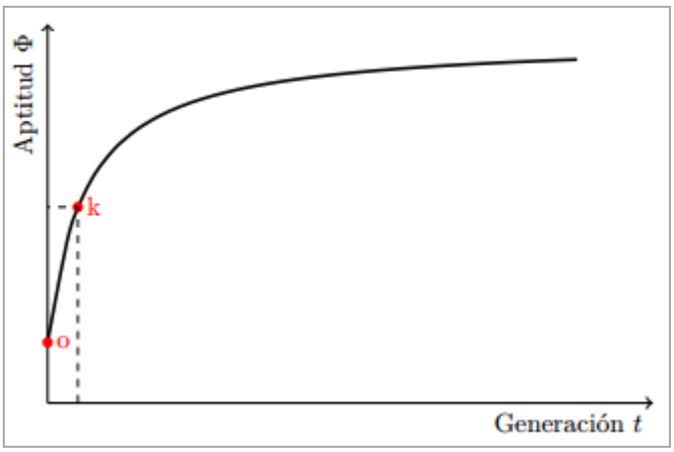

Figura 8 – Rendimiento de los Algoritmos Genéticos

Los criterios de detención más comunes son:

- El que los individuos hayan encontrado una estabilidad en cuando a su función de aptitud durante las *t* generaciones.
- El que la aptitud de los individuos ya no mejore en *t* generaciones llegando a un estancamiento del algoritmo.
- El número máximo de generaciones o tiempo que se viene ejecutando el algoritmo. Éste se aplicará cuando ninguno de los anteriores consiga alcanzarse.

2.2.4.5 La función de aptitud

La función de aptitud o también denominada función de *fitness*, es uno de los métodos centrales de los algoritmos genéticos, ya que de ella depende cuantificar la bondad de cada individuo, lo cual implica directamente que éste tenga más o menos opciones de reproducirse y permanecer en las siguientes poblaciones. En torno a este concepto surgen otros dos que son el *manejo de restricciones* y las *técnicas de escalado* que se describen a continuación.

El *manejo de restricciones* ha sido ampliamente estudiado y se han definido diferentes métodos dentro de los cuales las *funciones de penalización* son los más usados. Éstos fueron propuestos por Curant y ampliados por Carroll y por Fiacco y McCormick, y consiste en añadir

o restar un determinado valor a la función objetivo dependiendo del grado de violación de las restricciones. Se han desarrollado diferentes enfoques como la *penalización estática*, en los que los factores de penalización son determinísticos –se establecen a priori y permanecen constantes durante todo el proceso de optimización; la *penalización dinámica*, en donde los factores de penalización son dependientes del número de generaciones transcurridas incrementándose con el tiempo; la *penalización adaptativa*, que va cambiando –adaptándose– a la evolución del algoritmo a partir de diferentes medidas tomadas; la *penalización coevolutiva*, que va incorporando los parámetros de penalización en el propio proceso evolutivo que es el encargado de seleccionar los más adecuados; el *algoritmo genético segregado*, propuesto por Le Riche, que utiliza dos subpoblaciones y dos funciones de penalización a diferencia de los otros métodos; la *pena de muerte* que es el modo más sencillo y consiste en asignar una aptitud nula si se pretende maximizar o muy grande si se pretende minimizar cuando un individuo viola una restricción; y finalmente, la *penalización Fuzzy,* propuesta por Wu, que se basa en la lógica difusa para variar los coeficientes de penalización.

Los *algoritmos de reparación* también forman parte del *manejo de restricciones* y son útiles en problemas como la secuenciación de tareas o rutas. Consiste en *reparar* individuos no factibles que sustituirán, según una probabilidad determinada de antemano, al individuo original en la población.

En la aplicación de los algoritmos genéticos, la resolución del problema se ve condicionada por la elección del tipo de codificación, los operadores genéticos, los parámetros y la función de aptitud. Uno de los problemas más comunes de los algoritmos genéticos es que convergen muy rápido hacia un óptimo local si la aptitud de los individuos es demasiado diferente. Otro problema surge cuando el algoritmo converge y los individuos son muy próximos en términos de aptitud haciendo que la presión selectiva –definida como la cantidad de veces que un individuo pueda ser seleccionado– desaparezca. Para evitar o minimizar estos problemas surgen las *técnicas de escalado* clasificadas de acuerdo a Goldberg en *escalado lineal, truncado sigma* y *escalado potencial*.

El *escalado lineal* propone realizar una traslación y rotación de la función de aptitud. El *truncado sigma* propone emplear la desviación típica de la aptitud de la población para preprocesar la aptitud antes del escalado y el *escalado potencial* en la que la función de aptitud original es escalada por un exponente *k* cercano a la unidad, es muy común usar *k = 1,005*.

2.3 Proyectos Relacionados

Finalmente, se presentan algunos proyectos relacionados con el trabajo actual.

En 1994, Brian Arthur plantea el problema de "El Farol Bar". Un bar de Santa Fe, Nuevo México, en el que los jueves por la noche un grupo toca música Irlandesa, lo cual parece ser un atractivo para la población de Santa Fe que se siente muy motivada de visitar este bar; sin embargo, el bar tiene capacidad sólo para el 60% de la población y cuando se supera este límite la visita deja de ser agradable y las personas hubieran preferido no ir. Este problema ha sido estudiado por diversos investigadores que han intentado encontrar, con diversos métodos, la cantidad óptima de asistentes. En el trabajo realizado por William Rand y Forrest Stonedahl se han comparado las soluciones propuestas empleado cuatro mecanismos.

El primero de ellos emplea la aleatoriedad, así, cada persona lanzará una moneda que decidirá si irá o no al bar. Tomando en cuenta que el total de la población que puede ir al bar es de 100 personas, con la primera técnica empleada, en promedio sólo 50 personas irán al bar.

El segundo mecanismo empleado considera las técnicas de economía neoclásica, que sugiere que cada agente debe realizar su mayor esfuerzo para predecir la audiencia del bar llegando a dos resultados, (1) cada agente predecirá el mismo resultado, en este caso todos irán al bar o todos se quedarán en casa, o (2) debido a que existe un número infinito de formas de predecir el siguiente número en una secuencia finita, cada agente obtendrá una predicción diferente. Asumiendo que la mitad de las predicciones son que el bar estará lleno y la otra mitad que no lo estará, se puede decir que la mitad de los agentes *neoclásicos* irán al bar es decir la audiencia será también de 50 personas.

Arthur (1994) plantea el tercer mecanismo, en el que usa un grupo de estrategias que se asignan a cada agente y cada jueves usará la que mejor aplique a la decisión de la semana anterior. Siguiendo este proceso, se tendrá la máxima asistencia óptima de 60 personas. Arthur sugiere que resultados similares se obtendrían empleando algún algoritmo evolutivo dando lugar al cuarto mecanismo.

Fue Fogel et al. (1999) quien experimentó lo sugerido por Arthur usando un algoritmo evolutivo para cada agente con 10 estrategias que evolucionaban 10 generaciones cada semana. Sus resultados fueron que un promedio de 56 personas irían al bar.

En 1999, Bruce Edmonds publica su trabajo *Modelling Socially Intelligent Agents*, en el que trata el problema de "El Farol Bar" y menciona que hacer que comunidades de agentes muestren comportamientos caracterizados por agentes racionales socialmente inteligentes, es un paso hacia el modelado de algunos aspectos clave de los seres humanos en situaciones reales. Los agentes modelados por Edmonds presentan diversas limitaciones por ejemplo, son limitadamente racionales en varios aspectos, tienen memoria limitada y un límite en su capacidad de hacer inferencias a partir de sus modelos. Usa también una adaptación de la programación genética en la que el modelo interno de los agentes son un conjunto de expresiones de tipo árbol y la selección de una de ellas depende de que haya tenido éxito en el pasado o basados en mecanismos de aprobación. Así, cada agente tendrá una población de modelos mentales que corresponden con los modelos alternativos de su mundo compuestas

por dos expresiones: una que determina la acción (si ir o no) y otra que determina la comunicación con otros agentes. Los modelos internos serán generados aleatoriamente al principio, para luego ser evolucionados lentamente basados en las decisiones correctas del pasado o medidas de lo que antes ha tenido éxito. La Figura 9 ilustra la estructura del agente según Edmonds.

Cada "semana", se produce una nueva población de modelos mentales, de acuerdo con la programación genética, empleando un alto grado de propagación, por lo que pocos genes aleatorios nuevos se introducen cada semana. Estos modelos son evaluados de acuerdo a la función de *fitness,* ésta puede ser medida en base al éxito de las predicciones pasadas o de la utilidad que los agentes pueden haber ganado si usaban ese modelo en el pasado.

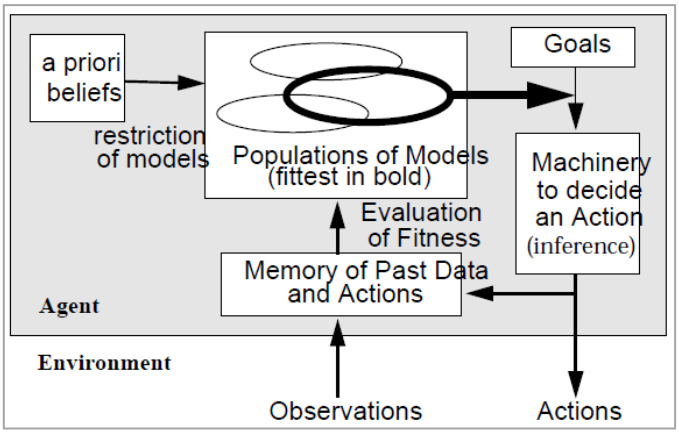

Figura 9 – Modelo de agente de Edmonds[1].

Se considera al trabajo de Edmonds como uno de los antecedentes del presente trabajo debido a que conjuga en un modelo social de agentes y el uso de técnicas de computación evolutivas.

Otro trabajo que está relacionado con el que se presenta en este documento, fue propuesto por Liu, Hong, Liu y Chai en 2011. Consiste en una nueva arquitectura para agentes virtuales en donde la prioridad de la motivación puede evolucionar. Ellos consideran que la motivación y personalidad son parámetros psicológicos para un agente virtual en donde la motivación es la causa directa de promover las emociones y comportamientos del agente. Las percepciones se reflejan principalmente en dos aspectos, el primero de ellos es que el agente virtual abstrae el impacto de los estímulos del entorno a sus propias propiedades después de la interacción, y el segundo es que el agente usa un módulo de memoria para extraer la evaluación pasada del estímulo del entorno. Este módulo de memoria incluye memoria de corto plazo y memoria de largo plazo. En condiciones reactivas, como el peligro, la información es enviada directamente el módulo emocional. El módulo de evaluación infiere la intensidad de la emoción por medio de lógica difusa. El módulo de comportamiento refleja los resultados de la evaluación, si se estimula una necesidad de motivación, se creará un comportamiento para la misma. El módulo que realiza las acciones o también llamado librería de acciones, ejecuta códigos de comportamiento. La Figura 10 ilustra esta arquitectura de agentes basada en motivaciones.

[1] Figura extraída de Edmonds, B. (1999) Modelling Socially Intelligent Agents.

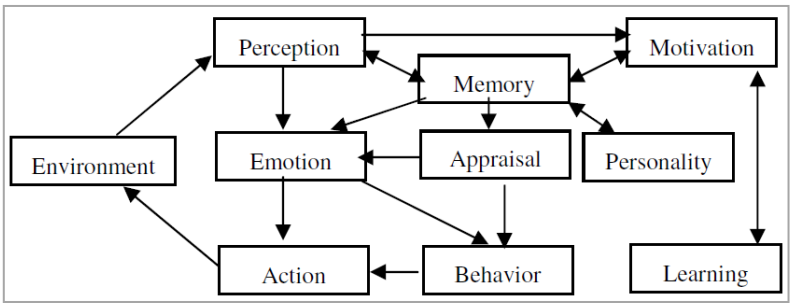

Figura 10 – Arquitectura del agente basado en motivaciones[2].

Los algoritmos genéticos se emplean en el establecimiento de prioridades de las motivaciones. Éstas están divididas en cinco categorías establecidas por Maslow que son las necesidades de sed, alimentación, seguridad, amor y curiosidad, y que pueden presentarse de forma simultánea. El algoritmo propuesto consiste en (1) codificación, que es el conjunto de niveles de prioridades de motivación que representan los genotipos; (2) generar la población inicial; (3) calcular la aptitud o *fitness* que representa la adaptación de la relación de precedencia del conjunto de motivaciones de un individuo. Este valor se establece por medio del *feedback* de los comportamientos, de este modo, si el resultado de los comportamientos encaja con el target inicial, obtiene un *feedback* positivo; de otro modo será uno negativo; (4) la selección, que consiste en obtener los individuos con menor grado de aptitud y reemplazar cada bit con algún bit de los otros individuos elegido aleatoriamente. Asegurando que el peor individuo es eliminado; (5) aplicar el cruzamiento o *crossover* que por cada individuo de la población se *resetean* cada uno de los bits usando los elementos correspondientes de otro individuo; y finalmente, (6) la mutación, que consiste en resetear cada bit del genotipo de cada individuo de acuerdo con la probabilidad. Si se obtiene un mejor genotipo de este proceso, éste deberá existir en el proceso evolutivo, de otro modo será eliminado con el tiempo. En el caso propuesto en su investigación, los autores demuestran que el uso de algoritmos genéticos ayuda a que los agentes se puedan adaptar mejor a su entorno. Como consecuencia de la aplicación de los algoritmos genéticos, si el entorno es estable, los agentes virtuales aprenderán como configurar su precedencia de motivaciones y podrán seleccionar la más apropiada.

Teniendo en cuenta lo anterior, el modelo emocional de los agentes virtuales indica que las emociones derivan de los objetos, agentes y eventos. Así, las emociones surgen de la evaluación de estos tres componentes y las variables motivacionales, que son parámetros importantes en el proceso. Por ejemplo, cuando un agente presenta la necesidad de alimentarse, debe lanzar la motivación de comer. Luego se determinará si la intensidad de la motivación es suficiente basándose en la experiencia pasada; si la motivación es generalmente satisfecha, se mostrará una emoción de esperanza, de otro modo aflorará una emoción de miedo o no se tendrá ninguna emoción. Una vez establecida la emoción, si el agente observa algún objeto puede juzgar si éste podría satisfacer su hambre basada en la experiencia pasada.

[2] Figura estraída de Liu, Z., Hong, Y., Lui, Q. y Chai, Y.J. (2011) An Emotion Model for Virtual Agents with Evolvable Motivation.

Al igual que el trabajo de Liu, Hong, Liu y Chai, esta investigación propone el uso de algoritmos genéticos como parte de los procesos internos del agente para generar algún tipo de conocimiento sobre el entorno y realizar las acciones que mejor se adecuen a él.

Sin embargo, el cometido de este trabajo es intentar averiguar el modelo personal de otros agentes simulando el comportamiento de las personas en el mundo real que llegan a "conocer" a otras personas basándose en su comportamiento. Acerca de ello no se han encontrado documentos relevantes que propongan alguna técnica o modelo que pueda aplicarse para conseguir este objetivo.

Capítulo 3

Planteamiento del Problema

3.1 Delimitación del Problema

A medida que se van mejorando las aplicaciones basadas en agentes, se siente más la necesidad de que éstos reflejen características humanas en su comportamiento, de modo que sean más creíbles y puedan cumplir mejor los objetivos para los cuales fueron diseñados. No se han hecho esperar variedad de investigaciones y trabajos realizados para incorporar estas características al agente consiguiendo que éste pueda mostrar expresiones faciales y corporales, emociones y comportamientos de gran similitud al de una persona, extendiéndose del mundo científico al entorno comercial introduciendo estas capacidades en aplicaciones ampliamente demandadas como son juegos, aplicaciones educativas basadas en agentes, aplicaciones orientadas a superar fobias y muchas otras.

Particularizando en el tema de emociones, los mayores desarrollos en aplicaciones comerciales se han realizado en el área de HCI (Human Computer Interaction), cuyo objetivo principal es mejorar la interacción y experiencias de los usuarios con las aplicaciones de ordenador. Aquí, se pretende identificar las emociones de los usuarios atendiendo, principalmente, a las características biométricas de los mismos como son los gestos, intensidad de la voz, calor corporal y todo cuando pueda ser capturado a través de dispositivos conectados al ordenador, con la finalidad de que los agentes puedan procesar y entender estas emociones para planificar acciones acordes con ellas. Estas acciones pueden abarcar desde mostrar uno u otro mensaje dependiendo de la emoción del usuario hasta la demostración de rasgos físicos emocionales en el agente. El éxito de estos avances han determinando un nuevo hito en el área de HCI, ya que el usuario se siente cada vez más identificado con estas aplicaciones informáticas aprovechándolas mejor y, por ende, mejorando considerablemente el cumplimiento de sus objetivos.

Enmarcados todavía en el entorno científico y más relacionado con el presente trabajo, surgen las arquitecturas emocionales de agentes que pretenden dotar al agente de un modelo personal que incluye las características definitorias del individuo, las emociones y las relaciones entre ellas. Esto surge para dar respuesta a la carencia de los modelos “tradicionales”, en los que se pretendía considerar solamente las variables racionales del

problema, ya que con ello se tenían resultados cada vez más exactos, de los que no se cuestiona su validez para aplicaciones industriales en las que no es relevante asemejar los comportamientos o la toma de decisiones a las de los seres vivos; sin embargo, no podrían aportar la gran riqueza que se desprende del modelo personal del individuo.

El planteamiento de estas arquitecturas emociones de agentes han sido el punto de partida para nuevas investigaciones orientadas a solventar los principales problemas que se han ido generando. Tal es el caso de la COGNITIVA (Imbert, 2005), que soluciona los problemas de generalidad de algunas arquitecturas, que dificulta enormemente su aplicación en problemas reales, y la especificidad de otras que no permiten que se las pueda aplicar a otros problemas, teniendo que redefinirlas por completo. Es así que COGNITIVA es una arquitectura que aporta los componentes necesarios para poder modelar diversos tipos de problemas que hacen uso de agentes cognitivos.

Teniendo como base a la arquitectura COGNITIVA, se ha identificado que aún hace falta desarrollar un modelo que permita al agente inferir los rasgos de personalidad de los demás agentes con los que interactúa, del mismo modo como las personas pueden identificar características personales de otras a medida que comparten experiencias entre ellas, para adaptar su comportamiento a éstos y maximizar su función de utilidad. Un ejemplo de ello son las conocidas situaciones en las que se tiene alguna necesidad y, dependiendo de la misma, se elige a alguien de nuestro entorno para solicitarle lo que se necesita; pero ¿cómo se conoce quién podría acceder a la petición?, o dicho de otro modo, ¿en función de qué se descarta solicitar la petición a ciertas personas? La respuesta a estos interrogantes es, que las personas almacenamos información sobre otros teniendo como base las experiencias previas que se haya compartido con los demás. Así, cuando se quiera pedir algún favor se recurrirá a quien se cree que es bondadoso o compasivo, o que se conozca que tiene mayor predisposición de ayudar a los demás. En la actualidad, no se ha encontrado planteamiento alguno documentado que pretenda inferir los rasgos de personalidad de otros agentes cognitivos para adaptar su comportamiento a ellos.

En este punto se hace necesario dar algunas definiciones que serán clave en el desarrollo de este proyecto. Éstos se detallan a continuación.

Agentes cognitivos: Como se ha descrito en el apartado dedicado a los agentes cognitivos, son aquellos capaces de razonar sobre la base de su conocimiento y poseen un estado mental que incluye creencias, conocimiento, comunicación, control y funcionalidad.

Modelo de inferencia: A efectos de este proyecto, se dirá que un modelo de inferencia es un grupo de elementos que, actuando de forma conjunta, es capaz de deducir ciertas características a partir del comportamiento de un individuo.

Características definitorias: Tal como su nombre indica, vienen a ser aquellos aspectos propios de recintos, objetos o individuos que las diferencian unos de otros y que determinan su comportamiento, según sea el caso. En el presente este trabajo, estos términos serán empleados como un sinónimo de los rasgos de personalidad, ya que se tratará principalmente con individuos.

Es así, que el propósito principal de esta investigación es conocer si el comportamiento del agente se adapta mejor a sus objetivos cuando éste considera las características definitorias de los demás agentes para tomar sus decisiones. Para ello, se dotará a los agentes de estructuras en las que pueda ir almacenando los rasgos de personalidad de los demás a medida que va "observando" su comportamiento, y que luego empleará cuando tenga que decidir sobre qué respuesta o acción tomar. De este modo, la pregunta de investigación que se pretende solucionar en este proyecto es la siguiente.

¿Se adapta mejor el comportamiento del agente cuando conoce las características definitorias de los demás agentes con los que interactúa?

3.2 Objetivos y Limitaciones

El objetivo principal es mostrar que los agentes pueden adaptar mejor su comportamiento cuando conocen las características definitorias de otros agentes.

Para cumplir con este objetivo se debe realizar lo siguiente:

- Crear un modelo que permita inferir las características definitorias de otros agentes.
- Probar el funcionamiento del modelo de inferencia creado aplicándolo a un caso de estudio.
- Determinar la ganancia que aporta el modelo de inferencia comparando los resultados que se obtienen de éste con otros, que sean producto de modelos que no usen los rasgos de personalidad.

Para el cumplimiento de estos objetivos se hará uso de una arquitectura reactiva de agentes en la que algunos agentes harán uso del modelo de inferencia propuesto para deducir las características definitorias de los demás, empleándolas en la toma de decisiones sobre las respuestas o acciones a realizar, mientras que otros emplearán otras técnicas a tal propósito.

La principal limitación que se ha encontrado en el desarrollo de este trabajo es la temporal, por lo que se ha tenido que reducir el alcance de la solución propuesta tal como se detallará en el siguiente capítulo.

3.3 Justificación

La idea de investigación fue recogida de las líneas de trabajo futuro de la tesis doctoral "Una Arquitectura Cognitiva Multinivel para Agentes con Comportamiento Influido por Características Individuales y Emociones, Propias y de Otros Agentes" (Imbert, 2005), en la que se menciona que la *inferencia del modelo personal de los otros* puede tener un alto grado de interés permitiendo al agente adaptar su actuación y basar sus decisiones en función de lo que él cree del resto de los individuos y cómo espera que se comporten. Este proyecto aporta un avance en el mundo académico ya que plantea un modo de abordar este problema abierto que a día de hoy no tiene una solución conocida.

La incorporación del modelo personal en el comportamiento del agente enriquece enormemente las aplicaciones en las que es necesario asemejar el comportamiento de los agentes al de los seres humanos, ya que es el único modo de simular reacciones distintas ante

los mismos estímulos y condiciones del entorno, consiguiendo que su comportamiento sea cada vez más creíble. Si además de ello, se dota al agente de un modelo que le permita inferir los rasgos de personalidad de los demás, se conseguirían resultados de mucho impacto en la simulación de sistemas multiagente, pudiendo ser usados en una gran diversidad de áreas como la educación, la salud, el entretenimiento, los sistemas de seguridad, etc.

Capítulo 4

Planteamiento de la Solución

El objetivo de este capítulo es describir la solución propuesta, que consistirá básicamente en una arquitectura cognitiva multiagente en la que algunos de los agentes estará dotado de un modelo de aprendizaje que le permitirá ir conociendo progresivamente las características definitorias de los demás agentes y adaptando su comportamiento a ellas. En los siguientes apartados se detalla cada uno de los componentes de esta solución.

4.1 Arquitectura de agentes

La arquitectura de agentes que se emplea en el modelo propuesto está basada en la arquitectura COGNITIVA descrita en el apartado 2.1.4, en la que el módulo cognitivo estará conformado solamente por el nivel reactivo.

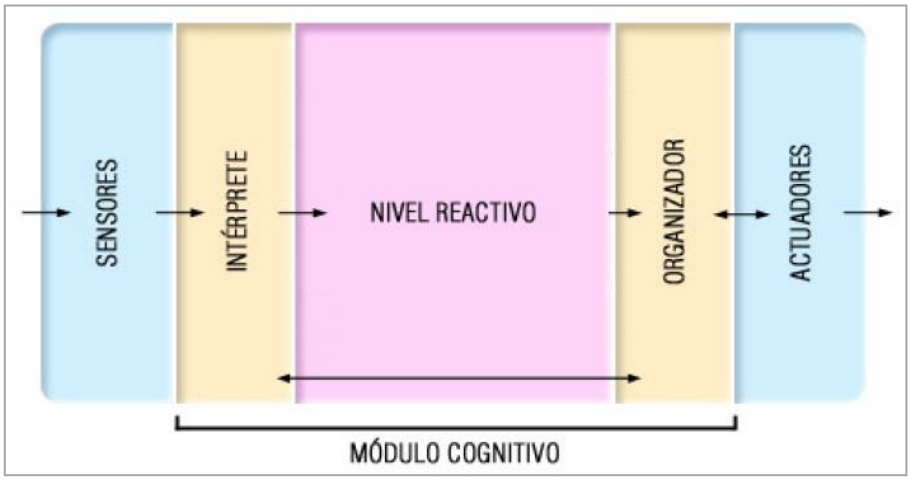

Figura 11 – Arquitectura de agentes propuesta[3].

Se ha considerado sólo el nivel reactivo debido a que éste aporta todos los componentes necesarios para cumplir con los objetivos del presente trabajo y permitir centrar la investigación en los mismos sin añadir mayor complejidad. Es importante hacer énfasis en que

[3] Imagen adaptada de Imbert, R. (2005) Una Arquitectura Cognitiva Multinivel para Agentes con Comportamiento Influido por Características Individuales y Emociones, Propias y de Otros Agentes.

una *reacción* no significa que se descarte lo que se cree o conoce de todo con lo que se haya interactuado, más bien todo lo contrario. La respuesta reactiva de los agentes tomará en cuenta las creencias que se tenga sobre sí mismo, los elementos del entorno y los demás agentes del sistema, que viene siendo el punto clave de este trabajo y permite que el módulo cognitivo sea simplificado al nivel reactivo. Aún así, se conoce que para modelar problemas reales son de vital importancia los niveles deliberativo y social por lo que, al igual que en COGNITIVA, pueden ser agregados en disposición horizontal sin invalidar la arquitectura actual.

A efectos de esta investigación, el modelo personal se ha reducido a tres componentes clave: los rasgos de personalidad, las emociones y las relaciones entre ellas. De este modo, y tomando la especificación definida en (Imbert, 2005), cada agente estará compuesto por una representación de sus propios rasgos de personalidad *P*, un conjunto de emociones *M* y las relaciones entre la ellas, así como una representación del modelo personal de los demás agentes, que tendrán los mismos componentes. Así, cuando el agente perciba un cambio y/o nueva información del entorno, ésta será procesada en el módulo cognitivo teniendo en cuenta su propia configuración y la de los demás agentes implicados para tomar una decisión sobre la acción a realizar.

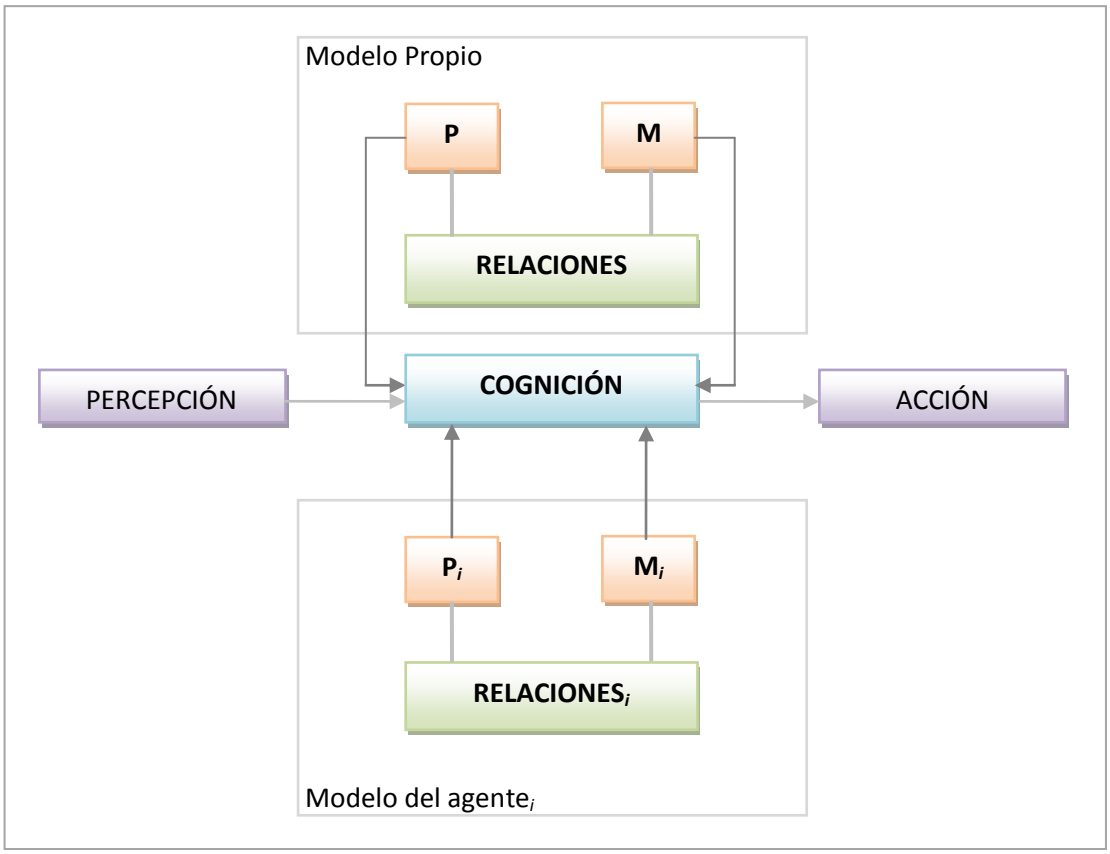

Figura 12 – Componentes del agente.

P viene a ser un conjunto finito de rasgos de personalidad del modelo personal del propio agente y se define como:

$$P = \{p_1, p_2, ..., p_n\} \quad (4.1)$$

Donde: $n \in N$ y es el número máximo de rasgos de personalidad considerados siendo el elemento p_n el rasgo de personalidad n del agente.

Para representar el conjunto de características definitorias de los demás agentes es necesario considerar, además de lo anterior, el número del individuo en mención:

$$P_i = \{p_{i,1}, p_{i,2}, ..., p_{i,n}\} \quad (4.2)$$

En el que $p_{i,n}$, $1 \leq n \leq |P_i|$ representa uno de los rasgos de personalidad del individuo i.

El valor de un determinado rasgo de personalidad viene dado por:

$$\pi = P \text{ -> } D_P \quad (4.3)$$

Donde D_P represente el dominio en el que se encuentran los valores de los elementos de P.

De manera similar se definen los valores de M que representa el conjunto de emociones o estados transitorios del agente. Así, para el propio agente se tiene:

$$M = \{m_1, m_2, ..., m_n\} \quad (4.4)$$

Donde: $n \in N$ y es el número máximo de emociones consideradas siendo m_n la emoción n del agente en un momento determinado.

Las emociones del individuo i se denotan por:

$$M_i = \{m_{i,1}, m_{i,2}, ..., m_{i,n}\} \quad (4.5)$$

Donde $m_{i,n}$, $1 \leq n \leq |M_i|$ se corresponde con una de las emociones del individuo i.

El valor concreto de una emoción en un determinado momento se encontrará dentro de su dominio definido y se representa de la siguiente manera:

$$\mu = M \text{ -> } D_m \quad (4.6)$$

En este trabajo se parte de la premisa de que las emociones se encuentran influenciadas por los rasgos de personalidad del individuo, consiguiendo con ello que, diferentes agentes tengan reacciones distintas ante las mismas percepciones. Además de los propios rasgos de personalidad, se deberá tener en cuenta el modo en el que éstos influirán (aumenta, disminuye, etc.) y el grado con el que dicho rasgo actúa sobre la emoción que deberá pertenecer a un dominio de valores determinado ($deg \in D_{deg}$).

El conjunto de influencias es también finito y se denota de la siguiente manera:

$$inf = \{inf_1, inf_2, ..., inf_n\} \quad (4.7)$$

Tanto los elementos de *P*, *M* y las relaciones entre ellos son totalmente dependientes del problema al que se aplicará el modelo, salvo por el hecho que deben estar comprendidos dentro del conjunto de los números naturales incluido el cero. En el siguiente capítulo se desarrolla un caso de estudio en el que se definirán cada uno de estos valores.

4.2 Modelo de inferencia

Un modelo de inferencia es un prototipo que hará uso de algún método o técnica para deducir algo a partir de las características dadas. En el presente trabajo será empleado con el objetivo de encontrar las características definitorias de otros agentes, teniendo como punto de partida la percepción de sus emociones. Dentro de la arquitectura de agentes propuesta, el modelo de inferencia estará incluido dentro del módulo cognitivo, permitiendo al agente ajustar sus respuestas reactivas a los rasgos de personalidad que ha conseguido inferir de interacciones anteriores y, de cumplirse la hipótesis propuesta, estas respuestas deberían adaptarse progresivamente en beneficio del agente que aplica este modelo.

Para desarrollar este modelo de inferencia, se planteó inicialmente un conjunto de escenarios de complejidad creciente que se enumeran a continuación:

1º Escenario

- El modelo personal es idéntico al de los demás:
  - Mismos rasgos de personalidad *P*.
  - Mismas emociones *M*.
  - Mismas relaciones entre ellos.
  - Mismo conjunto de percepciones y su interpretación.
  - Mismo conjunto de acciones y su determinación.
- Lo que diferencia a cada individuo y su actuación es el valor de sus rasgos de personalidad. El conjunto de emociones *M* es muy variable y dependiente del tiempo mientras que sus características definitorias *P* son más estables y se asume que no deberían cambiar.
- La pregunta sería: a partir del comportamiento del otro individuo ¿se podrían inferir sus valores de *P*? ¿Cómo? De este modo se conocería mejor cuál va a ser el comportamiento futuro del individuo en gran parte de los casos (no en todos debido que podría depender de más cosas como la historia pasada, las actitudes, los estados físicos, etc.)

2º Escenario

- El modelo personal del individuo es similar al de los demás, excepto en las relaciones entre *P* y *M*.
  - Mismos rasgos de personalidad.
  - Mismas emociones.
  - Mismo conjunto de percepciones y su interpretación.
  - Mismo conjunto de acciones y su determinación.
- La cuestión a resolver sería: a partir del comportamiento del otro individuo ¿es posible inferir los valores de *P* y las relaciones entre *P* y *M*?

3º Escenario

- El modelo personal del individuo es distinto al de los demás pero se comparte la percepción y actuación. No se conoce qué componentes tendría pero estaría basado en un modelo cognitivo con *P* y *M*.
    - Mismo conjunto de percepciones y su interpretación.
    - Mismo conjunto de acciones y su determinación.
- La pregunta sería: a partir del comportamiento del otro individuo ¿es posible inferir |P|, |M|, sus valores y las relaciones entre ellos?

4º Escenario

- Todo es distinto, incluido los modelos de percepción, actuación y las propias percepciones y acciones.
- Se intuye que sin un modelo de aprendizaje no será capaz de conocer qué es lo que percibe y cuál es su actuación para poder interpretarla y poder inferir todo lo demás. Esto aplicaría a un indígena de una selva por ejemplo, que viendo a alguien de una civilización más desarrollada utilizar un ordenador trataría de darse cuenta de qué es lo que mira, qué es lo que hace y por qué.

El alcance del presente trabajo incluye realizar sólo el primero de estos escenarios, dando la primera, y más sencilla, aproximación del modelo que puede ser extendido para ir cubriendo, paulatinamente, los demás. En este escenario, debido a que todos los componentes del modelo personal son los mismos, el objetivo del modelo se resumiría a encontrar los valores de los rasgos de personalidad de los otros agentes.

El modelo de inferencia propuesto hará uso de los algoritmos genéticos (descritos en el apartado 2.2), que es una de las técnicas de la computación evolutiva y presentan gran potencialidad tratando problemas de los que no se tiene un amplio conocimiento previo. En este caso en particular, se parte de la premisa de que los agentes no han interactuado antes y no conocen cómo podría ser el comportamiento de los demás. En una situación real, cuando una persona interactúa con otra que acaba de conocer, no puede saber qué acciones o respuestas realizará, teniendo que "adivinar" en función de suposiciones sobre lo que observa y, dependiendo de la respuesta que reciba, se creará una u otra percepción de la misma. Si este proceso se repite continuamente, se puede decir que una persona llega a "conocer" a otra y a tener un cierto grado de certeza al anticipar su comportamiento. Este proceso se puede trasladar casi de forma natural a un algoritmo genético que parte de una población de individuos o soluciones aleatoria para luego ir acercándose progresivamente al individuo que se adapte mejor al problema, que será la configuración de valores de los rasgos de personalidad de los demás agentes.

La primera cuestión que se presenta cuando se hace uso de los algoritmos genéticos es cómo representar a los individuos. En este trabajo se empleará la codificación binaria de los valores comprendidos dentro del dominio de los rasgos de personalidad. La longitud de la cadena binaria marcará entonces el límite superior de valores que pueden ser considerados. Por ejemplo, con un número binario de 4 bits se podrían representar 16 ($2^4 = 16$) valores distintos de cada rasgo.

Un cromosoma, entonces, estará compuesto por el número de bits elegido y se corresponderá con un rasgo de personalidad, es decir, si se pretende inferir los valores de 4 rasgos de personalidad, cada individuo de la población estará compuesto por 4 cromosomas, debido a que deben ser soluciones al problema, en el que cada uno de estos tendría un número de alelos igual a la longitud de la cadena binaria. De modo general se tiene que:

$$P_i = \{p_{i,1}, p_{i,2}, ..., p_{i,n}\} \tag{4.8}$$

Quedaría representado por:

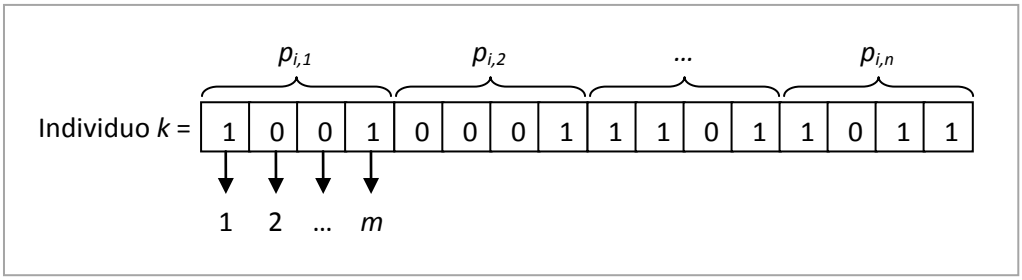

Figura 13 – Representación de un individuo del algoritmo genético.

Una vez representados los individuos de la población, se debe definir la función de adecuación o *fitness* que, generalmente, viene a ser una de las tareas más difíciles de la aplicación de los algoritmos genéticos, ya que de ella depende que se consiga encontrar individuos cada vez mejores. Además de ello, y también muy importante, se debe parametrizar los siguientes puntos:

1) Número de individuos de la población
2) Número de generaciones.
3) Modo de creación de la primera generación de individuos.
4) Tipo de selección.
5) Tipo de cruce.
6) Probabilidad de cruce.
7) Tipo de mutación.
8) Probabilidad de mutación.

Todos estos aspectos incluida la función de adecuación, son totalmente dependientes del problema a resolver por lo que a este nivel no pueden ser parametrizados.

Este modelo de inferencia será ejecutado cuando se requiera dar una respuesta en la que las características definitorias de quien la solicita juegan un papel primordial. El disparador de la ejecución del modelo será la recepción de información de los estados emocionales de los demás agentes, ya que partiendo de éstos se podrá inferir los rasgos de personalidad debido a la influencia de éstos, definida en el apartado anterior.

4.3 Validación del modelo

La finalidad principal de esta investigación es mostrar que, en arquitecturas cognitivas de agentes, el entender las características propias del modelo personal juega un papel muy importante en la toma de decisiones que impacta directamente en la maximización del beneficio de los agentes. Para ello, se plantea comparar el uso del modelo propuesto con otras técnicas que no tengan en cuenta los rasgos de personalidad de los agentes al momento de decir que acción emprender, de modo que se pueda saber en qué medida aporta el conocimiento de las características definitorias al problema a resolver.

Además de lo anterior, se deben tomar en cuenta otros aspectos al momento de evaluar este modelo. La Figura 14 ilustra estas dimensiones.

Figura 14 – Dimensiones para validar el modelo propuesto.

- ✓ Uso de recursos: En arquitecturas reactivas de agentes, el tiempo de respuesta es una de las características más relevantes y de mayor impacto, por lo que se debe prestar especial consideración a la complejidad y, por ende, los recursos que pueda emplear el modelo para llegar a inferir soluciones aceptables.

- ✓ Solución aceptable: A diferencia de otros métodos, las técnicas de la computación evolutiva, no siempre alcanzan la misma solución, aún cuando se tengan exactamente los mismos parámetros ejecución tras ejecución. Esto se debe a que tienen una base dependiente del azar, lo que hace que en algunas ocasiones se obtenga la respuesta correcta al problema y en otras no. De cualquier forma, de no alcanzar la solución exacta que se busca, se debe verificar que los individuos resultantes sean aceptables para el problema que se intenta resolver.

- ✓ Ganancia del agente: Finalmente se debe medir el impacto en la ganancia del agente cuando se emplea el modelo y verificar que este impacto es relevante para el problema.

Capítulo 5

Caso de Estudio

Con la finalidad de validar el modelo propuesto, éste ha sido aplicado a una aproximación muy simple del un juego muy extendido en el norte de España que se denomina *Kinito*. Su origen se remonta al siglo XVII y es considerado una "evolución" del póker jugando con dados. En la actualidad este juego se practica en País Vasco, Zamora, La Rioja, Burgos, Palencia, Valladolid, Navarra, Asturias, León y Cantabria.

En los siguientes apartados se comienza realizando una descripción de la aproximación de este juego que será empleada en el presente trabajo; se sigue con las concreciones funcional y contextual, que se propone en COGNITIVA y que son necesaria para la programación de los agentes en la plataforma JADE[4], en la que estará incluido el modelo propuesto dentro del módulo cognitivo de cada agente. Finalmente, se presentan los resultados de modo que se pueda mostrar la validez del modelo.

5.1 Descripción del Juego

A efectos de este trabajo, el juego consiste en que cada jugador, por turnos, deberá lanzar los dados para obtener un número que sólo él puede ver. Una vez conocido el número obtenido, deberá elegir de forma secuencial a otro jugador y le mencionará en voz alta el número que supuestamente ha obtenido (que puede ser el valor verdadero u otro que él crea conveniente). Por su parte, el jugador seleccionado deberá decidir si cree o no que el número es el verdadero y, dependiendo de ello, se penalizará a uno de los dos jugadores teniendo en cuenta las reglas que se definen en los siguientes apartados. El jugador penalizado deberá dar al jugador ganador un número de monedas, terminando así la partida y volviendo a comenzar otra hasta que alguno de los jugadores se haya quedado sin monedas o el número de partidas sea 300.

El objetivo de cada uno de los jugadores será apoderarse del máximo número de monedas.

5.1.1 Reglas del juego propuesto

- El juego comienza cuando se tiene como mínimo tres jugadores disponibles.
- Todos los jugadores comenzarán con el mismo un número monedas.

[4] Se describe JADE con mayor detalle en el Apéndice D o en http://jade.tilab.com/.

- El jugador que lanza los dados podrá decidir si decir el número verdadero obtenido, en tal caso se dirá que "No miente", o decir cualquier otro número superior al que se ha obtenido y menor o igual al máximo posible, diciendo en este caso que "Miente". Por ejemplo, si se obtiene el número 3, podrá decir 3 o cualquier otro número comprendido entre 4 y 6, ambos incluidos.
- El jugador seleccionado podrá decidir si "Cree" en el número enviado o no lo "Cree".
- Para determinar el jugador penalizado se tendrá en cuenta dos casos.
  1. Cuando no se alcance el máximo valor posible al lanzar los dados se tendrá en cuenta lo siguiente.

Jugador 1	Jugador 2	Jugador penalizado
No miente	Cree	Jugador 1
No miente	No cree	Jugador 2
Miente	Cree	Jugador 2
Miente	No cree	Jugador 1

Tabla 4 – Jugador penalizado cuando el número no es el máximo.

  2. Cuando se alcance el máximo valor posible al lanzar los dados se tendrá en cuenta lo siguiente.

Jugador 1	Jugador 2	Jugador penalizado
No miente	Cree	Jugador 1
No miente	No cree	Jugador 2

Tabla 5 – Jugador penalizado cuando el número es el máximo.

- El valor de la penalización será igual al número de monedas mencionado en voz alta por el jugador que lanza los dados.
- El juego termina cuando alguno de los jugadores se quede sin monedas o se alcanza la partida número 300.
- Gana el jugador que tenga más monedas.

5.2 Concreción de la Arquitectura COGNITIVA

Aunque la arquitectura COGNITIVA es aplicable en una gran variedad de problemas, presenta ciertos requerimientos mínimos que deben cumplirse para que pueda ser aplicada. A continuación se presentan estos requisitos y su justificación.

1. *Debe ser rico en interacción entre agentes*. El caso de estudio propuesto es rico en interacción entre los agentes a tal punto que no podría realizarse sólo con la participación un agente individual.
2. *Deben mostrar comportamientos influenciados por determinadas características personales explícitas*. Lo que se pretende en este caso de estudio es ir conociendo progresivamente la personalidad de los demás agentes, para lo cual es necesario que su comportamiento se vea influenciado por su modelo personal, en este caso específicamente los rasgos de personalidad. Además de ello, los agentes deberán comportarse de forma congruente con sus características personales.

3. *Los agentes deben mostrar una determinada influencia en su comportamiento debido a sus emociones*. Del mismo modo que el punto anterior, los agentes de este caso de estudio actuarán también en función de sus emociones.

Como se ha descrito en el apartado 2.1.4 dedicado a la Arquitectura Cognitiva del Agente –COGNITIVA–, se presentan dos tipos de concreción que se deben realizar para modelar sistemas de agentes cognitivos. En el trabajo realizado por Imbert, se ha desarrollado de forma muy completa y detalla una *concreción funcional* de carácter genérico que ha sido contextualizado en dos problemas con características distintas demostrando que es válido para una amplia variedad de problemas, por lo que será empleada también en este caso de estudio con algunas variaciones que se mencionan en el siguiente apartado.

5.2.1 Concreción funcional

Selección de los dominios y aritmética de operación

El dominio cualitativo por el que se ha optado es: <Muy poco, Poco, Medianamente, Mucho, Excesivamente>. El modelo semántico para las creencias del modelo personal en esta concreción funcional se muestra en la siguiente figura.

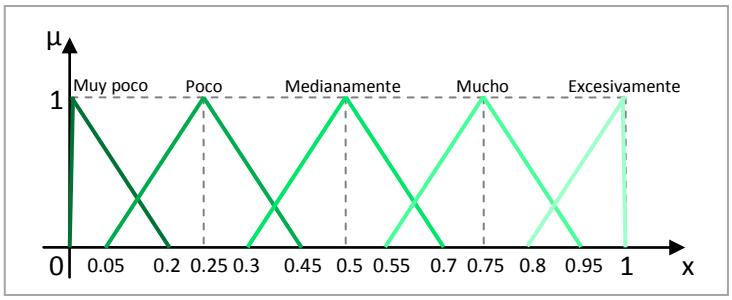

Figura 15 – Dominio cualitativo.

Teniendo en cuenta la notación de conjuntos borrosos, los atributos del modelo personal son:

<Muy poco>	$(0.001, 0.001, 0.199)_{LR}$
<Poco>	$(0.25, 0.2, 0.2)_{LR}$
<Medianamente>	$(0.5, 0.2, 0.2)_{LR}$
<Mucho>	$(0.75, 0.2, 0.2)_{LR}$
<Excesivamente>	$(0.999, 0.199, 0.001)_{LR}$

Caracterización de los Componentes de la Arquitectura

Del mismo modo que COGNITIVA, las creencias serán representadas como ternas *concepto-atributo-valor*, en las que el *concepto* hace referencia al individuo, objeto, recinto o situación; el *atributo* responde a la cualidad que se valora en la creencia; y el *valor* se corresponde con la valoración que se realiza de dicha cualidad (Imbert, 2005). Por su parte las relaciones entre creencias estarán compuestas por:

- *Influyente*: Atributo cuyo valor puede influir en el valor de otro atributo del modelo personal.
- *Influido*: Atributo sobre el cual el influyente ejerce su influencia.

- *Modo de Influencia*: Sentido en el que el influyente ejerce su influencia sobre el influido. Por ejemplo, el influyente podría hacer incrementar el valor del influido, disminuirlo, atenuarlo, etc. Los modos de influencia considerados en esta concreción son:

  AUMENTA: El influyente aumenta el valor del influido. Si se tiene X AUMENTA <deg>, el valor del atributo *Y* será:

  $$<Y>' = <Y> + <X> \otimes <deg> \quad (5.1)$$

  CAMBIA: Este modo de influencia está compuesto por el modo descrito anteriormente AUMENTA y el modo DISMINUYE que se representa de la siguiente forma:

  $$<Y>' = <Y> - <X> \otimes <deg> \quad (5.2)$$

  Así, cuando el valor de <X> se encuentra en el rango [0, 0.5] se aplicará el modo AUMENTA y cuando se encuentre en el rango (0.5, 1] se aplicará el modo de influencia DISMINUYE.

- *Grado de Influencia*: Intensidad con la que el influyente ejerce su influencia en el influido. El dominio cuantitativo denotado por D_{deg} está compuesto sólo por dos elementos:

  <medianamente, mucho>

  El modelo semántico correspondiente es:

<medianamente>	(0.5, 0.1, 0.1)LR
<mucho>	(0.75, 0.1, 0.01)LR

El esquema de una relación entre creencias del modelo personal del agente sería como sigue:

influyente MODO_de_INFLUENCIA <grado_de_influencia> influido

5.2 Concreción Contextual

Contextualización de las Creencias

Los tipos de creencias que serán contextualizadas son las referidas a *recintos*, a *objetos*, a *individuos* y a la *situación actual*. La definición de cada una de estas se ha detallado a lo largo del apartado 2.1.4.

Creencias acerca de Recintos: Este tipo de creencias no serán consideradas ya que el lugar en el que se encontrarán físicamente los agentes no influirá en su comportamiento. Incluso en las condiciones reales del juego, el recinto es bastante simple y no aporta gran influencia a los jugadores.

Creencias acerca de Objetos: Se ha identificado sólo un objeto significativo en esta concreción cuyas creencias se presentan en la Figura 16.

TABLE

Características Definitorios:

- identificador; nombre único para distinguir a cada mesa.

Estados Transitorios:

- estadoActual; determina el estado en el que se encuentra la mesa. estadoActual <pertenece> [inPlay, waitingToPlay]
- players; lista de jugadores (*identificador*$_{agente}$, *player*), corresponde a todos los jugadores –*players*– registrados.
- playerInPlay; identifica al jugador (*identificador*$_{agente}$, *player*) que se encuentra actualmente a cargo de la partida.

Figura 16 – Creencias acerca del objeto TABLE.

Creencias acerca de Individuos: Los individuos corresponden a los jugadores denominados *players*. Se destacarán creencias del modelo personal del individuo conformado por los *rasgos de personalidad* y *emociones*.

Los *rasgos de personalidad* que se han seleccionado para cada uno de los jugadores serán:

- *Tendencia a mentir*: Expresa la frecuencia en el que un jugador altera el resultado obtenido al lanzar los dados. Así, un jugador que tenga una tendencia a mentir alta modificará el resultado casi siempre.

TENDENCIA A MENTIR: (-) veraz → mentiroso (+)

- *Capacidad para fingir*: Determina la facilidad que tiene el jugador para alterar la expresión de sus emociones, de modo que, si se encontrara muy nervioso puede ocultarlo para que los demás jugadores no lo perciban.

CAPACIDAD PARA FINGIR: (-) transparente → engañoso (+)

- *Ambición*: Representa el deseo o anhelo de ganar más monedas.

AMBICIÓN: (-) desinteresado → codicioso (+)

La única *emoción* que se han considerado es el nerviosismo que expresa el estado de calma de un agente en un momento determinado.

NERVIOSISMO: (-) calma → intranquilidad (+)

Las actitudes no serán tomadas en cuenta en este caso de estudio para no agregar complejidad al modelo, y debido a que, el principal cometido de este trabajo es el proceso de aprendizaje del modelo personal de los demás jugadores.

De este modo, las creencias del agente jugador se resumen en la Figura 17.

PLAYER

Características Definitorios:

- identificador; nombre único para distinguir a cada jugador.
- P; conjunto de rasgos de personalidad del individuo.
  P = {capacidad de fingir, tendencia a mentir}

Estados Transitorios:

- M; conjunto de las emociones del individuo.
  M = {nerviosismo}
- monedasDisponibles; cantidad de monedas del jugador en un momento dado.
- estadoActual; estado del jugador, puede encontrarse a cargo de la partida actual, respondiendo a la solicitud de otro jugador o esperando a que llegue su turno de jugar. *estadoActual <pertenece> {inCharge, inResponse, waiting}*

Figura 17 – Creencias acerca del individuo PLAYER.

Creencias acerca de la Situación Actual: Que corresponde a las creencias de lo que está aconteciendo en cada instante en el entorno.

SITUACIÓN ACTUAL

- fase; fase en la que se encuentra el juego en un momento determinado.
  fase = {acciones del primer jugador, acciones del segundo jugador, establecimiento del ganador}

Figura 18 – Creencias acerca de la situación actual.

Para completar este apartado se debe especificar las relaciones entre las creencias del modelo personal, de modo que, se conozca cómo influirán los rasgos de personalidad en las emociones mostradas por los jugadores. Se menciona sólo los rasgos de personalidad y las emociones debido a que, no se tendrán en cuenta los estados físicos ni las actitudes. La Figura 19 resume estas relaciones.

RELACIONES ENTRE CREENCIAS

Influencia de los Rasgos de Personalidad sobre las Emociones:

capacidad de fingir CAMBIA <medianamente> nerviosismo
tendencia a mentir AUMENTA <medianamente> nerviosismo
ambición AUMENTA <mucho> nerviosismo

Figura 19 – Relaciones entre creencias.

Una vez más, como no se ha considerado estados físicos, no será necesario definir valores de reposo. Con respecto a los parámetros de degradado, se considerará que el jugador en cada partida comenzará con un grado de nerviosismo igual a cero.

Particularización de la Historia Pasada: En este punto se debe contextualizar los *sucesos* a ser recordados, las *creencias* del individuo que se verán afectadas y la *importancia* de cada uno de los sucesos considerados.

SUCESOS Y CREENCIAS PARA LA HISTORIA PASADA

EligeJugador(jugador):

(La mesa seleccionará al jugador a cargo de la partida cuyo identificador es *jugador*)

- $jugadorPartida_{sitActual} = jugador$
- $monedasDisponibles_{jugador} = monedas$
- $fase_{sitActual} = partida$

ObtieneNúmero(jugador, número):

(El jugador lanza los dados y obtiene un número)

- *númeroObtenido = número*

Juega(jugador, contrincante, número):

(El jugador elige un contrincante de tipo *jugador* y le dice el número que aparentemente ha obtenido)

- $jugadorPartida_{sitActual} = jugador$
- $monedasDisponibles_{jugador} = monedas$
- $jugadorSeleccionado_{sitActual} = contrincante$
- $monedasDisponibles_{contrincante} = monedas$
- *númeroMencionado = número*

Gana(jugador, número):

(Se revela el número real obtenido por el jugador y según eso se decide quién gana la partida)

- *númeroReal = número*
- *ganador = jugador*
- *monedasGanador = monedas + número*

FinPartida():

(Concluye la partida actual)

$fase_{sitActual} = siguiente\ partida$

Figura 20 – Sucesos y creencias para la historia pasada.

Especificación de las Acciones Ejecutables por el Agente: Estas acciones deben contener las *precondiciones* necesarias, el *operador* que se ejecutaría una vez cumplidas, las *consecuencias* esperadas, las *expectativas* de la acción y el *indicador de caducidad* que ayudaría al organizador y a los planificadores.

Para los jugadores se tiene:

LANZAR DADOS

Precondiciones:

- = ($fase_{sitActual}$, *playing*)

Operador:

- *lanzarDados()*

Consecuencias:

- = ($numeroObtenido_{jugador}$, *número*)

Indicador de Caducidad:

- *duraciónPartida*

Figura 21 – Acción LANZAR DADOS.

Esta acción del agente requiere que la fase actual sea *playing* (“en juego”) y producirá que el agente obtenga un número que estará vigente mientras dure la partida.

Selección de los Eventos y los Perceptos: La descripción de los eventos percibidos es la siguiente:

EVENTOS

- La mesa ha seleccionado al jugador para esta partida cuyo identificador es *id*.
- El jugador obtiene un número *númeroObtenido*.
- El jugador selecciona a su contrincante cuyo identificador es *id* y le informará el número que ha obtenido o cualquier otro.
- El contrincante decide si creer o no en el número que ha dicho el jugador y menciona su respuesta.
- Se informa al contrincante el número real obtenido.

Figura 22 – Eventos.

Los preceptos son:

PRECEPTOS

SiguienteJugador(id):
(La mesa anuncia el siguiente jugador a cargo de la partida)

ElecciónContrincante(id)
(El jugador elige a su contrincante y le dice el número obtenido)

ResultadoContrincante()
(El contrincante menciona si cree o no en el jugador)

ResultadoPartida()
(Se anuncia el ganador y se establece la penalización del perdedor)

Figura 23 – Perceptos.

Establecimiento de las Capacidades Reactivas del Agente: Los reflejos son:

LANZAR DADOS

Disparadores:
- *SiguienteJugador*

Justificadores:
- No conoce número

Acciones Respuesta:
- *lanzarDados()*

Figura 24 – Capacidades reactivas de LANZAR DADOS.

5.3 Implementación en JADE

Para implementar el caso de estudio se hará uso de JADE (*Java Agent DEvelopment Framework*). JADE es una plataforma de desarrollo de aplicaciones basadas en agentes en conformidad con las especificaciones FIPA[5] (*Fundation for Intelligent Physical Agents*).

Un agente en JADE tiene que cumplir con las siguientes características:

- Tiene un nombre único en el entorno de ejecución.
- Se implementa como un único hilo de ejecución (*single-threaded*).
- Tiene un método de inicio (*setup*), que sirve para inicializar el agente incluyendo instrucciones que especificarán la ontología a utilizar y los comportamientos asociados al agente, y otro de fin (*takeDown*), que sirve para liberar recursos antes de la eliminación del agente.
- En su implementación se define una clase interna por cada uno de los comportamientos asociados al agente.

Los comportamientos especifican tareas o servicios que realiza un agente para lograr sus objetivos. Los agentes están programados en base a sus comportamientos. La programación basada en comportamientos debe realizar los siguientes pasos:

1. Determinar qué debe ser capaz de hacer el agente.
2. Asociar cada funcionalidad con un comportamiento.
3. Escoger el tipo de comportamiento.
4. Dejar a JADE la tarea de que un solo comportamiento se esté ejecutando en cada instante (*scheduling*).

La comunicación entre agentes es fundamental para conseguir la potencia propia de los sistemas multiagente, ya que determina el comportamiento social de los agentes. Las "conversaciones" entre agentes en JADE se realizan empleando el lenguaje de comunicación entre agentes FIPA-ACL que emplea un mecanismo asíncrono en el que cada agente tiene una

[5] FIPA fue formada originalmente como una organización con sede en Suiza en 1996 para desarrollar estándares de software para agentes heterogéneos e interactivos y sistemas basado en agentes. Estos estándares fueron aceptados por IEEE en junio de 2005. http://www.fipa.org/

cola de mensajes entrantes de la cual el agente puede leer el primer mensaje de toda la cola o el primero de ellos que satisfaga un requisito. La cola de mensajes es única para cada agente y, por ende, compartida por todos los comportamientos.

5.3.1 Comportamientos

En este apartado se describen cada uno de los comportamientos de los agentes según el rol que desempeñan. Para su rápida identificación, se les ha asignado un código que está compuesto por la abreviatura de *behaviour* ("BEHAV") y un número secuencial según el orden en el que se llevarán a cabo.

BEHAV1 – Solicitar jugar: Cuando un agente cuyo rol sea "PLAYER" se registre en el sistema, deberá enviar una solicitud al agente con rol "TABLE" para ser incluido en el juego.

BEHAV1	
Agente	*Player i*
Rol	PLAYER
Tipo de comportamiento	OneShotBehaviour
Condición de activación	-

BEHAV2 – Procesar solicitud: El agente con rol "TABLE" deberá decidir si aceptar o no la solicitud de juego del agente *Player i* teniendo en cuenta si se está jugando o no en ese instante.

BEHAV2	
Agente	*Table*
Rol	TABLE
Tipo de comportamiento	CyclicBehaviour
Condición de activación	Recepción de mensaje MSG1

BEHAV3 – Iniciar juego: Una vez cumplido el requisito de que se tenga cuatro agentes "PLAYER" registrados, el agente "TABLE" deberá seleccionar por turnos al jugador (*Player i*) encargado de iniciar la partida.

BEHAV3	
Agente	*Table*
Rol	TABLE
Tipo de comportamiento	CyclicBehaviour
Condición de activación	Que haya iniciado el juego y el agente "TABLE no se encuentre esperando la respuesta de un agente "PLAYER"

BEHAV4 – Iniciar partida: El agente *Player i* deberá obtener un número como resultado de lanzar un dado, seleccionar a otro de los agentes "PLAYER" de forma aleatoria (*Player j*) y decidir si decir el número real obtenido o no.

BEHAV4	
Agente	*Player i*
Rol	PLAYER
Tipo de comportamiento	CyclicBehaviour
Condición de activación	Recepción de mensaje MSG2

BEHAV5 – Responder partida: El agente *Player j* deberá decidir si cree o no en el número enviado por el agente *Player i*.

BEHAV5	
Agente	*Player j*
Rol	PLAYER
Tipo de comportamiento	CyclicBehaviour
Condición de activación	Recepción de mensaje MSG3

BEHAV6 – Establecer ganador: El agente *Player i* determinará el ganador de la partida en base a la respuesta recibida y deberá actualizar el número de sus monedas.

BEHAV6	
Agente	*Player i*
Rol	PLAYER
Tipo de comportamiento	CyclicBehaviour
Condición de activación	Recepción de mensaje MSG4

BEHAV7 – Finalizar partida: El agente *Player j* deberá actualizar el número de sus monedas de acuerdo con la información que envía el *Player i* y finaliza la partida.

BEHAV7	
Agente	*Player j*
Rol	PLAYER
Tipo de comportamiento	CyclicBehaviour
Condición de activación	Recepción de mensaje MSG6

BEHAV8 – Actualizar monedas: El agente "TABLE" deberá actualizar el número de monedas del agente.

BEHAV8	
Agente	*Table*
Rol	TABLE
Tipo de comportamiento	CyclicBehaviour
Condición de activación	Recepción de mensaje MSG9

5.3.2 Mensajes

A continuación se detalla cada uno de los mensajes que activarán los comportamientos. Del mismo modo como se hizo para los éstos, el código que se usará para los mensajes es el literal "MSG" y número secuencial según el orden en el que los mensajes son enviados.

MSG1 – Solicitud para jugar: Cuando un agente de tipo "PLAYER" es iniciado, enviará un mensaje al agente de tipo "TABLE" para preguntar si puede entrar en el juego.

MSG1	
Agente Origen	*Player i*
Agente Destino	*Table*
Tipo (*Performative*)	PROPOSE
Contenido	PLAY(MONEY: <*número de monedas*>)
Ontología	Contener la palabra "PLAY"
Comportamiento Origen	BEHAV1
Comportamiento Destino	BEHAV2

MSG2 – Acepta solicitud: Si aún no se iniciado el juego, el agente "TABLE" aceptará la solicitud.

MSG2	
Agente Origen	*Table*
Agente Destino	*Player i*
Tipo (*Performative*)	ACCEPT_PROPOSAL
Contenido	OK
Ontología	Contener la palabra "OK"
Comportamiento Origen	BEHAV2
Comportamiento Destino	BEHAV3

MSG3 – Rechaza solicitud: Si el juego ya ha sido iniciado, el agente "TABLE" rechazará la solicitud.

MSG2	
Agente Origen	*Table*
Agente Destino	*Player i*
Tipo (*Performative*)	REJECT_PROPOSAL
Contenido	KO
Ontología	Contener la palabra "KO"
Comportamiento Origen	BEHAV2
Comportamiento Destino	Do_Delete

MSG4 – Inicio de partida: El agente "TABLE" enviará un mensaje al jugador seleccionado *Player i* para que inicie con la partida.

MSG4	
Agente Origen	*Table*
Agente Destino	*Player i*
Tipo (*Performative*)	INFORM
Contenido	START
Ontología	Contener la palabra "START"
Comportamiento Origen	BEHAV3
Comportamiento Destino	BEHAV4

MSG5 – Datos: El agente *Player i* enviará un mensaje al agente *Player j* que debe incluir el número obtenido supuesto y el valor del grado de nerviosismo que quiera mostrar en ese momento.

MSG5	
Agente Origen	*Player i*
Agente Destino	*Player j*
Tipo (*Performative*)	INFORM
Contenido	INITIAL(NUMBER: <*número obtenido supuesto*>, NERVOUSNESS: <*valor del nerviosismo a mostrar*>)
Ontología	Contener la palabra "INITIAL"
Comportamiento Origen	BEHAV4
Comportamiento Destino	BEHAV5

MSG6 – Respuesta: El agente *Player j* enviará un mensaje al agente *Player i* dando la respuesta a los datos enviados.

MSG6	
Agente Origen	*Player j*
Agente Destino	*Player i*
Tipo (*Performative*)	INFORM
Contenido	RESPONSE(VALUE: <*lo cree o no*>)
Ontología	Contener la palabra "RESPONSE"
Comportamiento Origen	BEHAV5
Comportamiento Destino	BEHAV6

MSG7 – Ganador: El agente *Player i* enviará un mensaje al agente *Player j* informando quien es el ganador de la partida.

MSG7	
Agente Origen	*Player i*
Agente Destino	*Player j*
Tipo (*Performative*)	INFORM
Contenido	WINNER(VALUE: <*Id Agente*>)
Ontología	Contener la palabra "WINNER"
Comportamiento Origen	BEHAV6
Comportamiento Destino	BEHAV7

MSG8 – Monedas: Los agentes *Player i, j* enviarán un mensaje al agente "TABLE" informando su número actualizado de monedas.

MSG8	
Agente Origen	*Player i, j*
Agente Destino	*Table*
Tipo (*Performative*)	INFORM
Contenido	COINS(VALUE: <*Nº monedas*>)
Ontología	Contener la palabra "COINS"
Comportamiento Origen	BEHAV6, BEHAV7
Comportamiento Destino	BEHAV7

MSG9 – Finaliza partida: El agente *Player j* enviará un mensaje al agente "TABLE" informando la finalización de la partida.

MSG9	
Agente Origen	*Player j*
Agente Destino	*Table*
Tipo (*Performative*)	INFORM
Contenido	END
Ontología	Contener la palabra “END”
Comportamiento Origen	BEHAV7
Comportamiento Destino	BEHAV1

5.3.3 Descripción del proceso general

A continuación se describe el proceso general del juego implementado en JADE. La Figura 25 ilustra este proceso.

I. El proceso de funcionamiento general comienza cuando se registra un agente *Table*, el cual se quedará esperando que agentes *Player* se conecten y soliciten jugar.
II. Los agentes *Player* se registran en el sistema y automáticamente solicitan jugar al agente *Table.*
III. Ante una solicitud de jugar, el agente *Table* tiene dos opciones:
    a) Aceptar la solicitud, que lo realiza cuando el juego todavía no haya iniciado.
    b) Rechazar la solicitud, mientras el juego está activo.
IV. El agente *Table* iniciará el juego cuando se haya alcanzado el número de jugadores configurado.
V. Una vez iniciado el juego, el agente *Table* selecciona por turnos secuenciales al agente a cargo de la partida denotado como *Agente i*.
VI. El *Agente i* realizará las siguientes acciones:
    i. Lanzar el dado y visualizar el número obtenido si informar del mismo a los demás.
    ii. Decidir si dice el número obtenido o algún otro superior al mismo basado en sus rasgos de personalidad.
    iii. Seleccionar al agente destino también de forma secuencial que se denotará por *Agente j*.
    iv. Informar el número *aparente* al *Agente j*.
VII. El *Agente j* tendrá que realizar lo siguiente:
    i. Decidir si cree o no en el número enviado por el *Agente i* basándose en la técnica configurada para el mismo. Las técnicas que se han considerado son: una puramente aleatoria, otra basada en reglas de acuerdo con los escenarios anteriores y otra que emplee el modelo de inferencia propuesto basado en algoritmos genéticos.
    ii. Informar al *Agente i* si cree o no que el número enviado es el valor obtenido.
VIII. En este punto el *Agente i* tiene las siguientes tareas:
    i. Determine el ganador de la partida.
    ii. Actualizar su número de monedas según el número que ha informado al *Agente j*.
    iii. Informar al *Agente j* sobre el ganador de la partida.
    iv. Informar a *Table* el nuevo número de monedas que tiene después de la partida.

IX. El *Agente j* debe realizar:
    i. Actualizar su número de monedas.
    ii. Informar a *Table* el nuevo número de monedas que tiene después de la partida.
    iii. Informar a *Table* sobre la culminación de la partida.
X. El agente Table deberá decidir, en base al número de monedas, si termina el juego o no.

Las tareas del IV al IX, ambas incluidas, deberán ejecutarse hasta que no termine el juego.

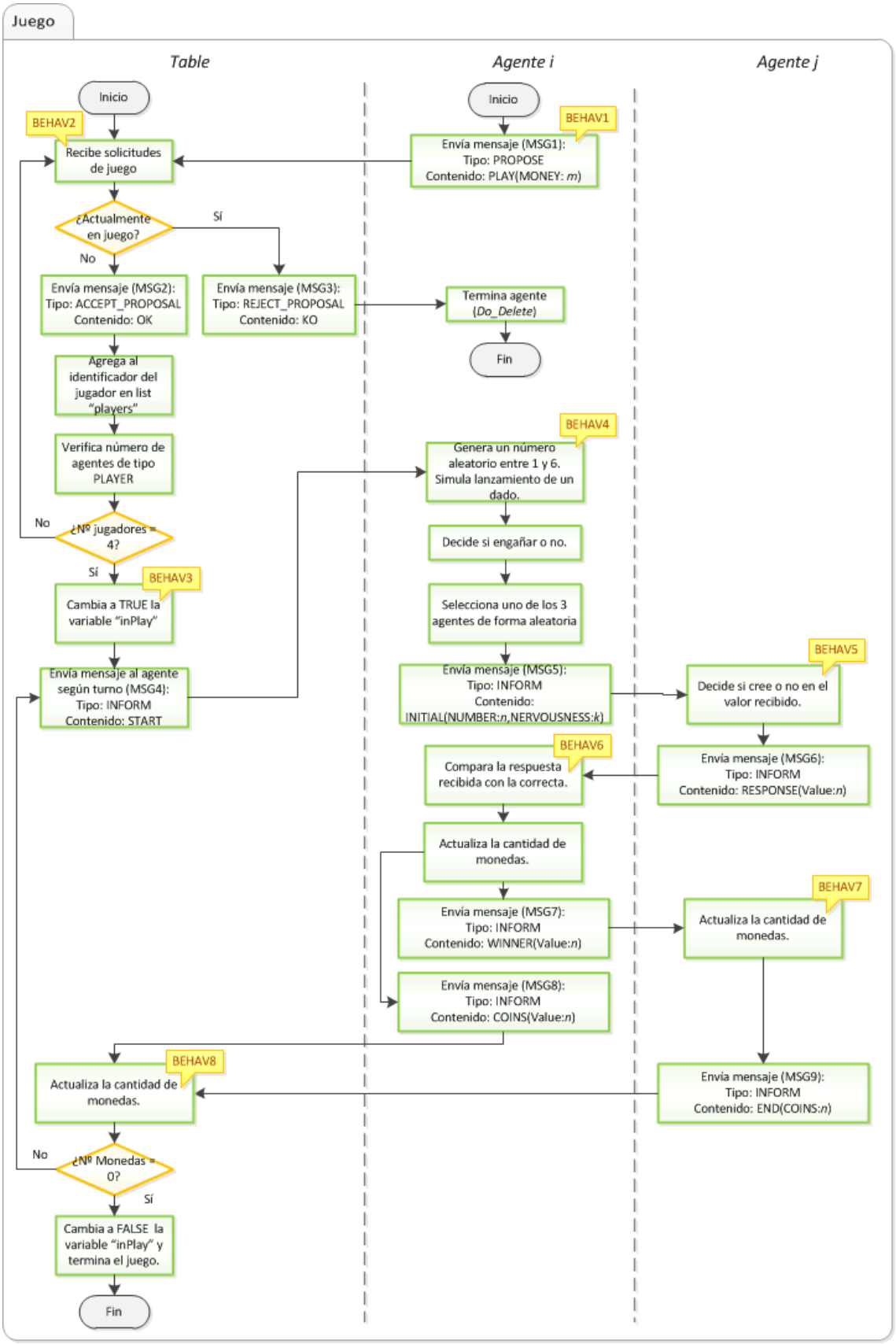

Figura 25 – Proceso general del juego.

5.4 Criterios de cálculo

Cálculo del nerviosismo según el número obtenido

Como ya se ha descrito en la concreción contextual para este caso de estudio, uno de los rasgos de personalidad considerados es la ambición que influye directamente en el nerviosismo del agente. De este modo, es lógico pensar que al obtener un valor muy bajo, como por ejemplo el número 1, el nerviosismo resultante para un agente excesivamente ambicioso debería ser el máximo valor posible; sin embargo si el agente es muy poco ambicioso apenas le importara el valor obtenido. Esto se refleja en la siguiente expresión:

$$m_1' = (6 - n)/5 * p_3 \qquad (5.3)$$

Donde: m_1' viene a ser el valor del nerviosismo inicial del agente y tiene una relación directa tanto con el complemento del número obtenido y el valor de la ambición del *Agente i*.

El resultado de esta fórmula se ilustra en la Figura 26.

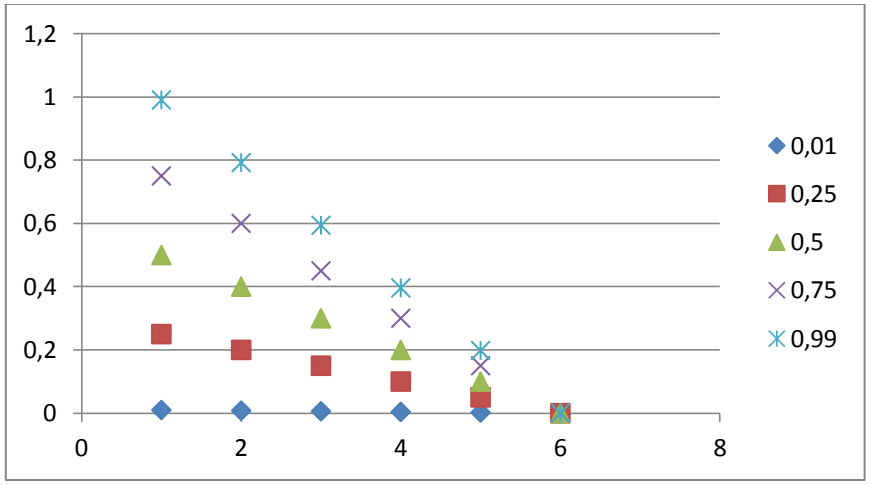

Figura 26 – Nerviosismo según el número obtenido.

Debido a que se ha elegido trabajar con valores discretos, estos resultados deben ser transformados teniendo en cuenta lo siguiente:

[0, 0.2)	0
[0.2, 0.4)	1
[0.4, 0.6)	2
[0.6, 0.8)	3
[0.8, 1)	4

Cálculo de la respuesta del *Agente i*

Una vez obtenido el número, el *Agente i* deberá decidir si elige decir el número verdadero (“No miente”) o incrementará su valor (“Miente”). Esta decisión está basada principalmente en el

primer rasgo de personalidad que viene a ser la tendencia para mentir y en menor medida por el nerviosismo calculado anteriormente. La expresión sería:

$$\%\ lie_{avg} = (p_1*0.8 + 0.2*m_1')*100 \quad (5.4)$$

Una vez calculado el promedio porcentual de veces que miente, se hallará un número aleatorio comprendido en el intervalo [1, 100] y de ser menor o igual que el valor encontrado, el *Agente i* elegirá mentir; de lo contrario dirá la verdad.

Cálculo del nerviosismo según la respuesta

El cálculo del nerviosismo que el *Agente i* mostrará será calculado teniendo en cuenta la respuesta que ha decidido dar ("Mentir" o "No mentir").

1º CASO - El *Agente i* ha elegido "Mentir": En este caso, el nerviosismo se calcula en dos fases secuenciales. La primera de ellas consiste en aplicar el valor del rasgo de personalidad p_1 que viene a ser la tendencia para mentir. Como se vio en el apartado correspondiente a la concreción contextual, el modo de influencia de este rasgo de personalidad sobre la emoción está definido por la función DISMINUYE de modo que el nerviosismo resultante quede representado por la siguiente expresión:

$$m_1'' = m_1' - p_1*deg_1 \quad (5.5)$$

Donde: m_1'' viene a ser el valor de la emoción después de aplicar p_1 en un grado de influencia deg_1 al nerviosismo inicial m_1'.

La segunda fase consiste en aplicar el rasgo de personalidad p_2 que viene a ser la capacidad para fingir cuyo modo de influencia sobre la emoción se corresponde con la función CAMBIA, que tal como se ha definido en la concreción funcional, será DISMINUYE cuando el valor de m_1'' es mayor o igual que 0.5 y AUMENTA cuando es menor. El resultado de esta fase será el nerviosismo que percibirán los demás agentes incluido el *Agente j*. Se representa de la siguiente manera.

$$m_1 = m_1'' \pm p_2*deg_2 \quad (5.6)$$

2º CASO - El *Agente i* ha elegido "No mentir": En caso de que haya decidido decir la verdad, el cálculo del nerviosismo se reduce sólo a la segunda fase del caso anterior debido a que cuando decide "No mentir", no tiene sentido aplicar su tendencia para mentir. La expresión quedaría como:

$$m_1 = m_1' \pm p_2*deg_2 \quad (5.7)$$

Donde: m_1' es el valor del nerviosismo inicial al cual se le aplica el valor de p_2 en un grado de influencia deg_2.

Cálculo del número según la respuesta

En caso de que el *Agente i* haya decidido mentir, debe mencionar un número superior al que ha obtenido que se calcula en base a su ambición ya que será el valor de la ganancia en caso de que este gane la partida de modo que mientras más ambicioso mayor debe ser el número que elija. Este valor se calcula con la siguiente expresión:

$$n' = n + p_3 * 1.2 \tag{5.8}$$

Donde: n' será la suma del número obtenido más el 120% del valor de su rasgo de personalidad p_3.

Cálculo de la respuesta del *Agente j*

La respuesta del *Agente j* puede tomar una de dos opciones. Si él considera, en base a la técnica empleada, que el número mencionado por el *Agente i* es verdadero, entonces le dirá que lo "Cree". De lo contrario si "piensa" que el *Agente i* ha enviado un número superior dirá que "No lo cree".

De acuerdo al Planteamiento de la Solución, para calcular este valor se tienen tres opciones que se describen a continuación:

a) Puramente aleatoria: Empleando esta técnica se generará un número aleatorio en el intervalo de [0, 100]; si este valor es menor a 50, la respuesta será que "No lo cree" en caso contrario será que lo "Cree".

b) Reglas: Este método hará uso de la información de la que dispone el agente en un momento dado. Esta información será almacenada en modo de tabla y ordenada según el número de partida creciente como la que se muestra en la Tabla 6:

Nº Partida	*n Agente i*	Respuesta *Agente j*	Respuesta *Agente i*
1	5	0 (No le cree)	0 (Miente)
2	…	…	…

Tabla 6 – Estructura de la tabla de información del agente.

Las reglas que se aplicarán a estos datos son:

- Rachas: Se seleccionará una cantidad impar de los últimos registros (como máximo cinco) y la respuesta que se haya dado menos será la que se dará en la partida actual. En la primera partida se seleccionará un valor aleatoriamente.
- Medias: Si la media de los números (como máximo 10 últimos) que haya dicho el *Agente i* es mayor a 3.5 se asumirá que el *Agente i* miente, de lo contrario dice la verdad.
- Tamaño: Si el número que ha mencionado el *Agente i* es mayor a 3.5 se asumirá que el *Agente j* miente, de lo contrario dice la verdad.

- Acumulado: Se calculará las veces que haya mentido y haya dicho la verdad por número y se seleccionará la opción que tenga un mayor valor para el número mencionado por el *Agente i*.
- Repeticiones: Respuesta del *Agente j* que más aciertos haya tenido.

c) Modelo de inferencia propuesto: Si se hace uso del modelo de inferencia propuesto, una vez encontrado el valor de adecuación de cada individuo de la población tanto para el caso de que el *Agente i* miente como para cuando no (estos cálculos se describen en el siguiente apartado), se procederá a sumar estos valores para cada una de las dos opciones y la respuesta será la que haya obtenido un menor valor de esta sumatoria. Por ejemplo, si la sumatoria de los valores de adecuación para el caso de que miente es 7 y lo propio para el caso de que no miente es 3, la respuesta será de que lo "Cree".

5.5 Configuración del Algoritmo Genético

En este apartado se darán a conocer los valores de configuración que empleará el algoritmo genético para este caso de estudio. Debido a que no existe un método exacto para calcular estos valores, se han calculado de forma empírica.

Número generaciones

El número de generaciones obtenido de forma empírica para este caso de estudio es de 100.

Número de individuos de la población

El espacio de búsqueda representa la cantidad de individuos posibles que pueden ser seleccionados como parte de la población. Para este caso, al tener tres rasgos de personalidad y cada uno tener cinco posibles valores de acuerdo con la definición del dominio cualitativo, el número total de individuos es:

$$5^3 = 125 \; individuos$$

Después de realizar varias pruebas se ha determinado que el valor óptimo de la población es de 5 individuos.

Representación del individuo

Como se mencionó anteriormente, un individuo estará formado por 3 cromosomas que corresponden a cada uno de los rasgos de personalidad teniendo como valor mínimo el número 0 y 4 como el máximo representando así a los cinco posibles valores de cada rasgo de personalidad.

Para representar un máximo de 4 se requiere de 3 bits de modo que los individuos serán:

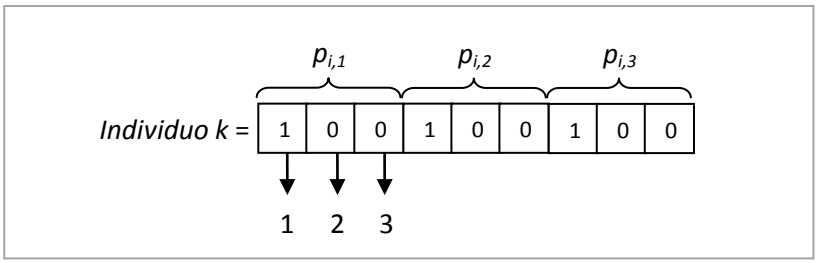

Figura 27 – Representación del individuo para este caso de estudio.

Debido a que empleando 3 bits se puede conseguir representar hasta 8 valores (2^3), los valores superiores al número 4 serán considerados *ilegales* por lo que cuando se genere uno de estos valores, ya sea producto de la primera generación o de a aplicación de los operadores genéticos, serán reemplazados por el máximo valor 4.

Modo de creación de la primera generación de individuos

Para la primera generación de individuos se ha convenido emplear valores aleatorios para los diferentes rasgos de personalidad ya que cuando inicia el juego, el agente no tiene ninguna información inicial sobre las características definitorias de los demás y por ende no se puede dar mayor o menor a las zonas del espacio de búsqueda.

Tipo de selección

En este caso de estudio se empleará la selección por torneo determinista con subgrupos de dos individuos. Se ha optado por este tipo de selección debido a que se intenta disminuir la presión de selección de modo que los individuos menos aptos también tengan la oportunidad de ser elegidos permitiendo la exploración de nuevas áreas en el espacio de búsqueda.

Tipo de cruce

El tipo elegido para el operador de cruce o *crossover* es el cruce por un punto debido a su simpleza ya que cuando más puntos de cruce se agreguen, menor será el rendimiento del algoritmo genético.

Probabilidad de cruce

La probabilidad de cruce que se ha elegido es del 0.7 y será aplicado de modo que se genere un número aleatorio en el intervalo [0, 1], el cual será comparado con esta probabilidad y si es menor o igual a 0.7 entonces se aplicará el operador de cruce, de otro modo se copiarán los padres seleccionados sin ninguna modificación.

Tipo de mutación

Para realizar la mutación de los individuos, se ha elegido la técnica de *multibit* en la que cada bit tiene la opción de mutarse o no. Se ha considerado esta opción para una mejor exploración del espacio de búsqueda.

Probabilidad de mutación

De modo general, en el uso de algoritmos genéticos, la probabilidad de mutación suele ser muy baja. Esto se debe a que, según demostraciones empíricas, el resultado de la función de adecuación de estos individuos es mucho menor a la de los padres pero sirve para asegurar que no queden áreas del espacio sin explorar. Para este caso de estudio se empleará el valor 0.08.

Mecanismo de reemplazo

El mecanismo de reemplazo de los individuos de una generación que se emplea para este caso de estudio hará uso del concepto de elitismo para mantener a los mejores individuos según el resultado de su función de adecuación. El porcentaje de elitismo considerado es del 20% del total de la población inicial.

Se incluyen también el 20% de individuos que hayan mutado y el 60% restante será completado con individuos resultantes del operador de cruce.

Función de Adecuación

Definir la función de adecuación o *fitness* es una de las tareas que más tiempo requiere cuando se aplica algoritmos genéticos ya que de ésta depende que los individuos vayan mejorando generación tras generación de modo que se pueda encontrar soluciones aceptables al problema.

Tal como se ha mencionado en el capítulo correspondiente al Planteamiento de la Solución, el alcance de este trabajo es realizar sólo la primera iteración propuesta de modo que los componentes del modelo personal sean los mismos para todos los agentes. De acuerdo con ello, el razonamiento aplicado es el siguiente: *cada uno de los individuos de la población inicial representarán configuraciones de los tres rasgos de personalidad que se han considerado, por ejemplo alguno de estos individuos puede tener una excesiva tendencia para mentir (cuyo valor sería la cadena binaria "100"), ser medianamente hábil fingiendo ("001") y ser muy poco ambicioso ("000"). Con esta combinación, lo primero que debería realizar el Agente j es "pensar" qué respuesta hubiera dado él mismo si tuviera esta configuración de sus características definitorias. Debido a que con esto aún no se puede conocer a ciencia cierta si el Agente i ha decido mentir o no, el Agente j debería evaluar ambos casos en términos de nerviosismo y compararlo con el que él percibe de modo que pueda tener una aproximación más clara de la respuesta que debe dar*.

Esta lógica es la se empleará como función de adecuación teniendo como parámetros de entrada el número mencionado por el *Agente i* y el nerviosismo que el *Agente j* perciba del *Agente i*. A continuación se describen los pasos para calcular el valor de adecuación de un individuo:

1º Encontrar el número obtenido de acuerdo con los valores de su ambición (p_3) y del número mencionado por el *Agente i*. Esto se realiza despejando el valor de *n* de la expresión (5.8), lo cual da como resultado:

$$n = n' - p_3 * 1.2 \quad (5.9)$$

2º Hallar el valor del nerviosismo inicial que tendría el *Agente i* de haber obtenido el número *n*. Para esto se emplea la expresión (5.3).

3º Calcular el valor del nerviosismo del *Agente i* percibido por el *Agente j* tanto para el caso de que haya decido mentir (m_{lie}) como de que no (m_{noLie}).

4º Debido a que el objetivo es encontrar los individuos más parecidos a la configuración real del *Agente i*, se comparará el nerviosismo percibido con los valores de nerviosismo calculados en el punto anterior restando dichos valores y aplicando el valor absoluto sobre el resultado que vendría a ser el valor de adecuación del individuo.

$$f(m_{lie}) = |m_{1,i} - m_{lie}| \quad (5.10)$$
$$f(m_{noLie}) = |m_{1,i} - m_{noLie}| \quad (5.11)$$

5.6 Resultados

Para mostrar los resultados de este caso de estudio se tomará como base las tres dimensiones consideradas en el apartado 4.3. En las siguientes líneas se analizará cada uno de estos aspectos de acuerdo con los resultados obtenidos de un total 10 ejecuciones del juego con tres agentes cuya configuración es la siguiente:

Agente	P1	P2	P3	Técnica
Player1	4 (100)	4 (100)	3 (011)	Modelo de Inferencia
Player2	3 (011)	0 (000)	2 (010)	Información disponible
Player3	4 (100)	0 (000)	0 (000)	Aleatoria

Tabla 7 – Agentes empleados.

En el que los valores de *P* son:

Excesivamente = 4
Mucho = 3
Medianamente = 2
Poco = 1
Muy poco = 0

Uso de recursos

El uso de recursos juega un papel importante en soluciones que se aplican a situaciones complejas. Los que tienen mayor impacto son, sin duda, el tiempo de procesamiento y el uso de memoria. Al ser éste un caso de estudio muy sencillo, los recursos computacionales no se han visto impactados significativamente; sin embargo se realizará una comparación de los tres métodos empleados en términos de tiempo.
Sin duda la técnica puramente aleatoria es la que requiere de menos tiempo para dar una respuesta ya que sería como “lanzar una moneda al aire y decir cara o cruz”.
El segundo método empleado basado en reglas, al requerir que éstas sean calculadas de acuerdo a la información del agente, tanto histórica como actual, necesitará un mayor tiempo dependiendo del volumen de información almacenada.

Estas dos técnicas no implican un aprendizaje, por lo que sólo se realiza una valoración del tiempo en responder a un estímulo del entorno, que en este caso, es que el agente a cargo de la partida lo seleccione como oponente y solicite su respuesta. Sin embargo; para el método de inferencia propuesto en este trabajo además del tiempo que tarde en encontrar una respuesta, que no tiene un impacto considerable debido que no ejecuta el algoritmo genético completo en cada partida del juego, sino que cada una de éstas es una iteración de dicho algoritmo, se tiene que valorar el tiempo que ha requerido para encontrar individuos que tengan un valor de adecuación aceptable. A continuación se describe una de las ejecuciones de este juego, en la que se podrá apreciar la curva de aprendizaje del algoritmo genético.

La población inicial generada aleatoriamente se muestra en la Tabla 8, en la que también se puede ver los valores de adecuación para cada uno de los individuos. Como se ha mencionado anteriormente, mientras menor sea el valor de adecuación, más parecido será el individuo a la configuración real de las características definitorias del agente.

Nº	Individuo	Valores de *P*	$f(m_{lie})$	$f(m_{noLie})$
1	000001000	P_i = {0, 1, 0}	4	0
2	001000100	P_i = {1, 0, 4}	0	1
3	100100100	P_i = {4, 4, 4}	0	1
4	100001000	P_i = {4, 1, 0}	1	1
5	100010011	P_i = {4, 2, 3}	2	2

Tabla 8 – Población inicial aleatoria.

Las primeras generaciones son bastante inestables y se corresponden con la fase de mayor aprendizaje del algoritmo genético. En este caso, es alrededor de la generación 24, que se obtienen individuos más parecidos a los valores reales (ver Tabla 9).

Nº	Individuo	Valores de *P*	$f(m_{lie})$	$f(m_{noLie})$
1	100000011	P_i = {4, 0, 3}	1	1
2	100000011	P_i = {4, 0, 3}	1	1
3	000000011	P_i = {0, 0, 3}	2	1
4	100000011	P_i = {4, 0, 3}	1	1
5	100000011	P_i = {4, 0, 3}	1	1

Tabla 9 – Población de la generación Nº 24.

La generación en la que se obtiene el valor exacto de los rasgos de personalidad del *Agente* i_1 en esta ejecución del juego, es la número 49. Los resultados se muestran en la Tabla 10.

Nº	Individuo	Valores de *P*	$f(m_{lie})$	$f(m_{noLie})$
1	011000010	P_i = {3, 0, 2}	0	0
2	011000010	P_i = {3, 0, 2}	0	0
3	011000010	P_i = {3, 0, 2}	0	0
4	011000010	P_i = {3, 0, 2}	0	0
5	011000010	P_i = {3, 0, 2}	0	0

Tabla 10 – Población de la generación Nº 49.

La Figura 28 ilustra el comportamiento de la función de *fitness* de esta ejecución del juego. Debido a que se calculan dos valores de adecuación, para el caso de que el agente mienta y para el caso en que diga la verdad, se ha representado ambos en ésta gráfica. Sin embargo, la función de *fitness* considerada es la mejor de ambas que se representa con la línea verde y que se denomina *fitness* general. Además de ello, se ha trazado una curva que representa la tendencia de la función de *fitness* general.

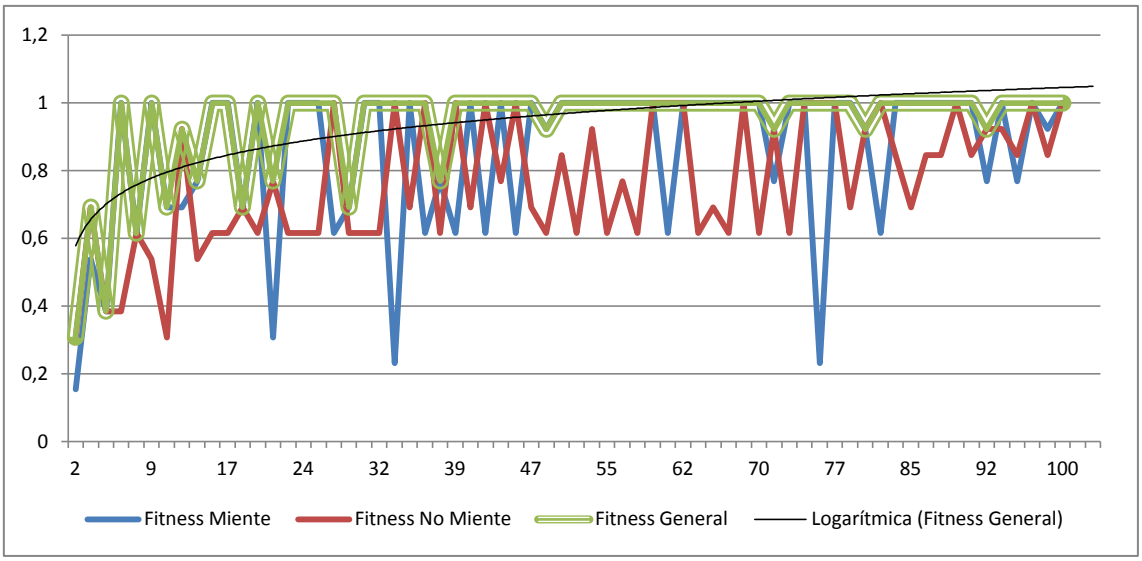

Figura 28 – Gráfica de la función de fitness de esta ejecución.

Para esta ejecución, el algoritmo genético ha tardado 49 generaciones en obtener los rasgos de personalidad reales del *Agente i*, pudiendo emplear esta información en su beneficio en las siguientes partidas. En la Figura 29, se ilustra la tendencia de la función de adecuación general de las 10 ejecuciones del juego. Se ha encontrado que el promedio de generaciones que requiere el algoritmo genético para encontrar soluciones aceptables es de 57.

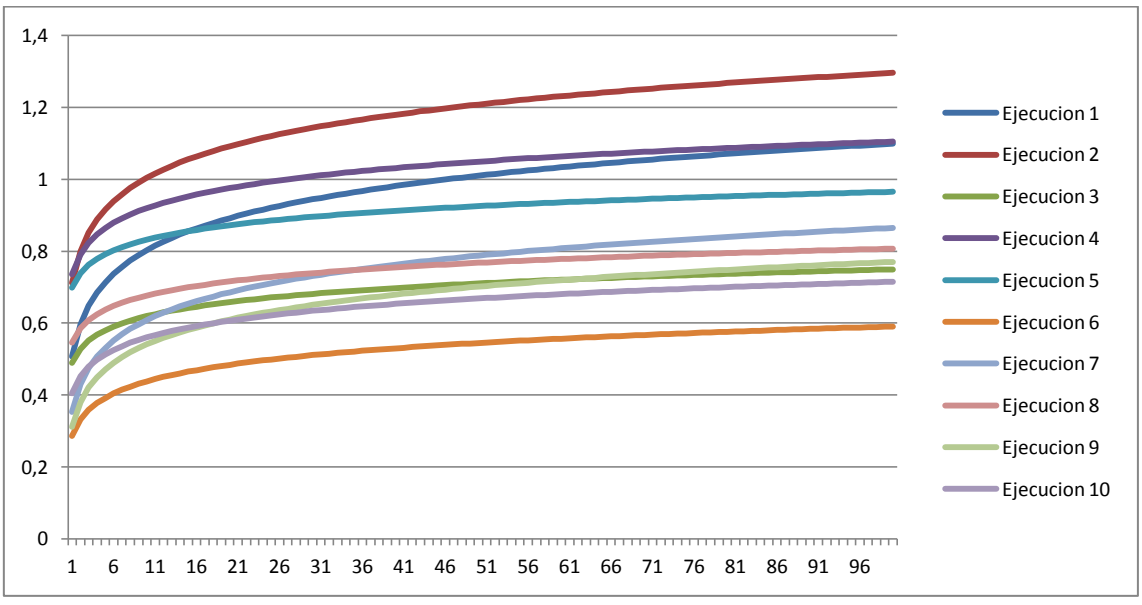

Figura 29 – Tendencia de la función de fitness general de las 10 ejecuciones.

Solución aceptable

Debido a que de los tres métodos empleados en este estudio, sólo el modelo de inferencia propuesto intenta llegar a conocer las características definitorias de los demás, esta dimensión estará dedicada solamente al análisis de los resultados del modelo de inferencia que se muestran en la Tabla 11. Las soluciones de color verde se consideran aceptables; mientras que las de rojo no.

Juego	*Agente i_1*			*Agente i_2*		
	p_1	p_2	p_3	p_1	p_2	p_3
1	010	000	010	100	000	001
2	010	000	010	011	001	000
3	001	000	010	100	000	000
4	011	000	010	100	000	000
5	100	000	010	100	000	000
6	011	001	001	011	000	000
7	011	000	010	100	000	000
8	011	000	010	100	001	000
9	011	000	010	100	000	000
10	011	000	010	100	000	000

Tabla 11 – Soluciones encontradas en las ejecuciones del juego.

De las 10 ejecuciones del juego, el agente que empleó el modelo de inferencia obtuvo los rasgos de personalidad exactos de los otros dos agentes en 4 ocasiones; sólo del primer contrincante pero no del segundo en 1 ocasión; sólo del segundo pero no del primero en 2 ocasiones y no encontró el valor exacto de ninguno en 3 ocasiones.

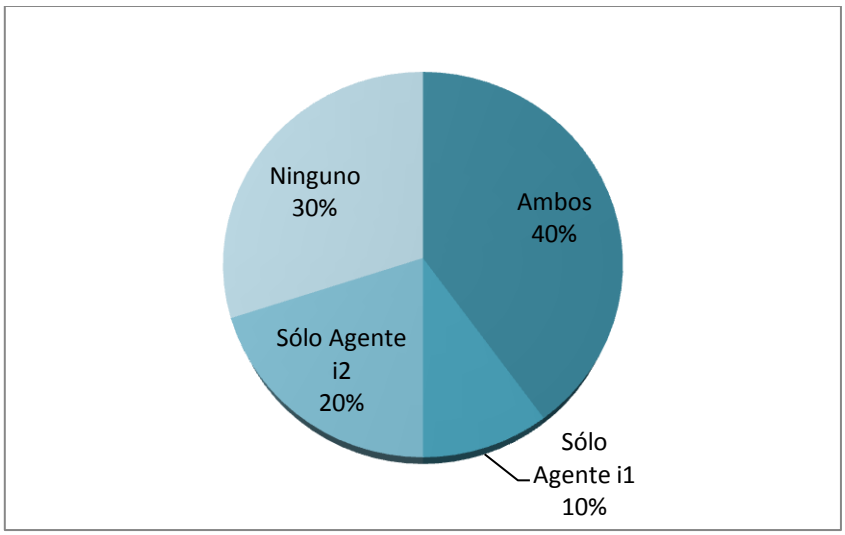

Figura 30 – Distribución de las soluciones encontradas.

Una solución será considerada aceptable cuando se cumplan dos condiciones: (i) sólo uno de los rasgos de personalidad sea diferente de la configuración del agente y (ii) esta variación sea de un solo elemento dentro del dominio cualitativo. De acuerdo con esto se tiene:

Juego Nº 1

La configuración obtenida por el modelo para *el Agente i_1 es P'* = {2, 0, 2} que significa que el agente tiene una tendencia para mentir media, su habilidad para fingir es muy poca y es medianamente ambicioso. El rasgo de personalidad que cambia es el primero, la tendencia para mentir, y lo hace en un solo elemento.

Con respecto al modelo encontrado para el *Agente i_2*, *P'* = {4, 0, 1}, varía también en un elemento del dominio cualitativo del rasgo de personalidad "ambición", pasando de ser muy poco ambicioso a poco ambicioso.

Se puede considerar entonces a los resultados del primer juego como aceptables.

Juego Nº 2

La configuración obtenida por el modelo para *el Agente i_1 es P'* = {2, 0, 2} que significa que el agente tiene una tendencia para mentir media, su habilidad para fingir es muy poca y es medianamente ambicioso. El rasgo de personalidad que cambia es el primero, la tendencia para mentir, y lo hace en un solo elemento. Esta solución cumple con los criterios para considerarse aceptable.

Con respecto al modelo encontrado para el *Agente i_2*, *P'* = {3, 1, 0}, varía en dos elementos del dominio cualitativo de los rasgo de personalidad "tendencia para mentir" y "habilidad para fingir", pasando de mentir excesivamente a hacer mucho y de tener muy poca habilidad para fingir a tener poca. Debido a que dos de los rasgos de personalidad cambian, no se considera una solución aceptable.

Juego Nº 3

La configuración obtenida por el modelo para *el Agente i_1 es P'* = {1, 0, 2} que significa que el agente tiene poca tendencia para mentir, su habilidad para fingir es muy poca y es medianamente ambicioso. El rasgo de personalidad que cambia es el primero, la tendencia para mentir, y lo hace en dos elementos, pasando de mucha a poca. Esta solución no es aceptable.

Con respecto al modelo encontrado para el *Agente i_2*, *P'* = {4, 0, 0}, corresponde a los valores reales del *Agente i*.

Juego Nº 5

La configuración obtenida por el modelo para *el Agente i_1 es P'* = {4, 0, 2} que significa que el agente tiene excesiva tendencia para mentir, su habilidad para fingir es muy poca y es medianamente ambicioso. El rasgo de personalidad que cambia es el primero, la tendencia para mentir, pasando de excesiva a mucha, por lo que se considera aceptable.

Con respecto al modelo encontrado para el *Agente i_2*, *P'* = {4, 0, 0}, corresponde a los valores reales del *Agente i*.

Juego Nº 6

La configuración obtenida por el modelo para *el Agente i_1 es P'* = {3, 1, 1} que significa que el agente tiene mucha tendencia para mentir, su habilidad para fingir es poca y es poco ambicioso. Los rasgos de personalidad que cambian son la habilidad para fingir, de muy poca a poca, y la ambición, de medio a poco. No es considerada una solución aceptable.

Con respecto al modelo encontrado para el *Agente* i_2, P' = {3, 0, 0}, sólo cambia en la tendencia para mentir de excesiva a mucha, pudiéndose considerar aceptable.

Juego Nº 8
La configuración obtenida por el modelo para *el Agente* i_1 *es* P' = {3, 0, 2} que viene a ser el valor real del *Agente i*.
Con respecto al modelo encontrado para el *Agente* i_2, P' = {4, 1, 0}, cambia en el valor de la habilidad para fingir, pasando de muy poca a poca y pudiéndose considerar aceptable.

Ganancia del agente

Finalmente, la tercera dimensión considerada para validar el modelo, se encuentra relacionada con la ganancia del agente, que se encontrará acorde con el problema.
Quizás esta dimensión sea la más importante de las 3, debido a que permite mostrar de un modo tangible, que el hecho de conocer las características definitorias de los demás, permite adaptar el comportamiento de los agentes para maximizar sus objetivos. Es más, esta dimensión indica claramente, si se conseguirá una ganancia considerable empleando el modelo en cualquier problema al cual sea aplicado, que es lo que más importa cuando se consideran situaciones reales.

Volviendo a este caso de estudio, el objetivo del juego era obtener el máximo número de monedas. El resultado de las 10 ejecuciones del programa se resume en la Tabla 12. De acuerdo con esta información, se tiene que el primer jugador, que es el que empleó el modelo de inferencia propuesto, ha ganado un 60% de las veces y ha obtenido en total 108 monedas adicionales; el segundo jugador, que empleaba información histórica, ha ganado el 30% de las veces terminando con 26 monedas extra; por último, el tercer jugador, que empleó la aleatoriedad sólo ha ganado el 10% de las veces y ha perdido 134 monedas.

Juego	Monedas Player 1	Monedas Player 2	Monedas Player 3
1	14	4	-18
2	-17	54	-37
3	-32	9	23
4	70	-44	-26
5	19	-1	-18
6	-72	71	1
7	56	-95	39
8	-34	33	1
9	40	37	-77
10	64	-42	-22
TOTAL	108	26	-134

Tabla 12 – Ganancia de los agentes.

Capítulo 6

Conclusiones

6.1 Modelo de Inferencia

El modelo de inferencia propuesto en este trabajo, se ha limitado a cubrir los requerimientos del primer escenario planteado como parte de la solución. En este escenario, el modelo personal del agente se ha reducido a tres componentes principales: los rasgos de personalidad, las emociones y las relaciones entre ambos. Se considera que éstos son los mismos para todos los agentes.

A efectos de esta investigación, la expresión "características definitorias" es usada para referirse a los rasgos de personalidad, retirando los demás componentes para no agregar complejidad al modelo.

Este modelo de inferencia ha sido agregado al módulo cognitivo del agente, el cual consiste, básicamente, en un nivel reactivo que recibe información de los sensores y envía acciones a los actuadores. Esto se traduce en una arquitectura reactiva de agentes, suficiente para esta primera aproximación, ya que los comportamientos reactivos también toman en cuenta las creencias del agente sobre lo que le rodea, incluyendo otros agentes.

El proceso de inferencia de este modelo se ha llevado a cabo haciendo uso de algoritmos genéticos, que son una herramienta potente para obtener soluciones a problemas de los que no se tenga conocimiento previo. De la aplicación de los algoritmos genéticos se puede extraer lo siguiente:

- No existe un método concreto para establecer los valores de configuración, como son: número de individuos, número de generaciones, tipo de selección, tipos y probabilidades de cruce y mutación y método de reemplazo. Estos valores deben ser hallados de forma empírica, lo cual hace que su configuración requiera de mucho esfuerzo.
- La definición de la función de adecuación o *fitness* es uno de los aspectos más críticos de la aplicación de algoritmos genéticos, ya que se debe encontrar un modo de medir la aptitud de un individuo, que será lo que le de continuidad generación tras generación.

- Una vez configurados y encontrada la función de aptitud, los algoritmos genéticos han generado resultados bastante buenos, mostrando que el uso de esta técnica evolutiva es adecuada para inferir características propias de un individuo a partir de su comportamiento.

6.2 Pruebas

Se ha conseguido probar el modelo de inferencia desarrollado en este trabajo, aplicándolo a una simplificación de un juego muy conocido en el norte de España, que es denominado *Kinito*. Aunque no es una situación que revolucione alguna de las áreas en las que se haga uso de sistemas de agentes, se considera que poder probar el modelo es un paso muy importante, ya que, a día de hoy, no se conoce de algún modelo o aplicación que intente inferir las características definitorias de otros en base a su conocimiento, siendo la aproximación de este juego una herramienta rica en interacción y que nos permite medir el resultado final obtenido.

La aproximación de este juego consiste en que un individuo, elegido por turnos, lance un dado y obtenga un número que sólo él puede ver, debiendo decidir si le dirá a su oponente el número real o algún otro que sea superior al obtenido. El oponente es elegido también por turnos y deberá responder si cree o no, que el número mencionado es el real. Si el oponente consigue adivinar, ganará la partida y se apoderará de un número de monedas igual al que número que ha mencionado el jugador inicial; caso contrario, es el otro jugador el que gana la partida y se apodere de las monedas.

Las características definitorias considerados para los agentes son: la tendencia para mentir, la capacidad para fingir y la ambición. La única emoción considerada es el nerviosismo, que se ve influenciado por los rasgos de personalidad.

Para poder realizar las pruebas se ha desarrollado un sistema de agentes en JADE, que es una plataforma de desarrollo de agentes basada en Java y que implementa los estándares FIPA.

Se han empleado tres agentes, de modo que cada uno de ellos emplee una técnica distinta para su toma de decisiones. Entre estas técnicas se ha considerado:

- Una totalmente aleatoria, que es la más simple, ya que consiste en obtener un número aleatorio entre [1..100] y si este valor es menor o igual a 50, se dará una respuesta; mientras que si es mayor, se dirá otra. Esta técnica es la menos recursos computacionales consume, pero la que ha generado los peores resultados.
- Una técnica basada en reglas aplicadas a la información disponible del agente, cuyo uso de recursos depende del volumen de información del que disponga y sus resultados han sido mejores que la anterior. El uso de recursos de esta técnica no ha sido relevante para este caso y se considera que dependerá del volumen de información que tenga que ser analizada.
- El modelo de inferencia propuesto, para el que el algoritmo genético ha conseguido individuos significativos, en promedio, en la generación 57. Los individuos resultantes han sido generalmente aceptables, es decir, iguales o muy parecidos a los valores reales de los rasgos de personalidad del agente y ha sido el método que ha tenido los mejores resultados.

6.3 Ganancia

La ganancia que ha conseguido el agente que ha empleado el modelo de inferencia propuesto, es muy superior a la de los demás métodos considerados. En términos de monedas, en un total de 10 ejecuciones del juego, se han conseguido ganar aproximadamente 4 veces más que la técnica basada en la información del agente, y ni qué decir de la técnica aleatoria, que ha tenido que pagar la ganancia de los demás agentes.

La **conclusión final** que se puede obtener del presente trabajo es que infiriendo las características definitorias de otros agentes el comportamiento del agente se vuelve más eficiente porque no actúa a ciegas hacia los otros, sino que anticipa sus comportamientos, lo que es capaz de manejar en su beneficio.

Capítulo 7

Líneas de Trabajo Futuro

Como en la mayoría de investigaciones, tanto al plantear el alcance como a medida que se va desarrollando cada uno de los apartados van surgiendo temas que por diversos motivos no pueden ser abordados pero que contribuirían grandemente a completar y/o mejorar el presente trabajo. Se han identificado los siguientes puntos:

Completar las iteraciones propuestas en la solución, ya que en este trabajo sólo se ha abordado la primera de ellas ya que se pretendía ilustrar el funcionamiento del modelo con la iteración más sencilla de todas para luego extenderlo a las de mayor complejidad.

Probar el modelo propuesto con otros casos de estudio. Sería de especial relevancia aplicar este modelo en otros casos de mayor impacto en el entorno científico o en situaciones de mayor utilidad en el mundo real.

Integrar un mecanismo de aprendizaje para los agentes que "observan" la partida. Similar al comportamiento de las personas que no sólo se aprende el modo de ser de los demás cuando se tiene experiencias directas con ellos sino también cuando se observa su comportamiento, se podría incluir algún mecanismo para actualizar el modelo personal de los demás a medida que se va observando una partida en la que el agente no participe directamente.

Desarrollar un módulo gráfico para la solución, aunque esto es muy dependiente del caso a resolver se podría integrar el modelo en un módulo gráfico que permita ver el funcionamiento e interacción con los agentes con mayor detalle.

Mejorar y completar el estado de la cuestión, que debido a las limitaciones temporales sea abordado de forma incipiente haciendo falta por ejemplo: comparaciones entre diferentes soluciones existentes no sólo en agentes sino en cualquier rama de la informática en las que se intente conocer configuraciones internas de los demás componentes de una misma arquitectura; una comparación entre las diversas arquitecturas emocionales de agentes.

Bibliografía

[1]. Arthur, B. (1994) Inductive Reasoning and Bounded Rationality (The El Farol Problem). Stanford University and Santa Fe Institute.

[2]. Baker, J.E. (1985) Adaptive selection methods for genetic algorithms. In Proceedings of the 1st International Conference on Genetic Algorithms, Hillsdale, NJ, USA.

[3]. Blickle, T. y Thiele, L. (1996) A comparison of selection schemes used in evolutionary algorithms. Evolutionary computation, 4(4).

[4]. Booker, L. (1982) Intelligent Behavior as a Adaptation to the Task Environment. PhD thesis, University of Michigan.

[5]. Bratman, M. E. (1987) Intentions, Plans, and Practical Reason. Harvard University Press, Cambridge, MA.

[6]. Bratman, M. E., Israel, D. y Pollack, M. (1988) Plans and Resource-Bounded Practical Reasoning. Computational Intelligence, 4(4).

[7]. Brooks, R. A. (1991) Intelligence without Representation. Artificial Intelligence.

[8]. Brustoloni, J. C. (1991) Autonomous Agents: Characterization and Requirements. Carnegie Mellon Unviersity.

[9]. Buckles, B. P. y Petry, F. E. (1992) Genetic Algorithms. IEEE Computer Society Press.

[10]. Davis, D. N. y Lewis, S.J. (2003) Computational Models of Emotion for Autonomy and Reasoning. Informatica (Special Edition on Perception and Emotion Based Reasoning).

[11]. Davis, L. (1991) Handbook of Genetic Algorithms. Van Nostrand Reinhold.

[12]. Darwin, C. (1866) The origin of species by means of natural selection, or the preservation of favoured races in the struggle for life. J. Murray, London.

[13]. De Jong, K.A. y Sarma, J. (1992) Generation gaps revisited. In Foundations of Genetic Algorithms. Morgan Kaufmann.

[14]. Díaz, M. (2010) Genetic algorithms in the minimization of functions on small intervals. Departamento de Geoinformática. Universidad de las Ciencias Informáticas. Geysed.

[15]. Edmonds, B. (1999) Modelling Socially Intelligent Agents. Centre for Policy Modelling. Manchester Metropolitan University.

[16]. Edmonds, B. (2000) Gossip, Sexual Recombination and the El Farol bar: modelling the emergence of heterogeneity. Journal of Artificial Societies and Social Simulation.

[17]. Franklin, S. y Graesser, A. (1996) Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Institute for Intelligent Systems. University of Memphis.

[18]. Fogel, D.B., Bäck T. y Michalewicz Z. (2000) Evolutionary computation. Vol. 1, Basic algorithms and operators. Institute of Physics Pub., Bristol; Piladelphia.

[19]. Fogel, L., Owens, A. y Walsh, M. (1965) Artificial intelligence through a simulation of evolution. Biophysics and Cybernetic Systems, Spartan, Washington DC.

[20]. Goldberg, D.E. (1989) Genetic Algorithms in Search, Optimization and Machine Learning. The University of Alabama. Addison-Wesley, Reading, MA.

[21]. Haupt, R. L. y Haupt, S.E. (2004) Practical genetic algorithms. John Wiley, Hoboken, N.J.

[22]. Hayes-Roth, B. (1995) An Architecture for Adaptive Intelligent Systems. Artificial Intelligence: Special Issue on Agents and Interactivity.

[23]. Hewitt, C. E. (1977) Viewing Control Structures as Patterns of Passing Messages. Artificial Intelligence, 8(3).

[24]. Holland J. H. (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press.

[25]. Imbert, R. (2005) Una Arquitectura Cognitiva Multinivel para Agentes con Comportamiento Influido por Características Individuales y Emociones, Propias y de Otros Agentes. Tesis Doctoral, Universidad Politécnica de Madrid.

[26]. Koza, J. R. (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge.

[27]. Liu, Z., Hong, Y., Lui, Q. y Chai, Y.J. (2011) An Emotion Model for Virtual Agents with Evolvable Motivation. Faculty of Information Science and Technology. Ningbo University.

[28]. Maes, P. (1995) Artificial Life Meets Entertainment: Life like Autonomous Agents. Communications of the ACM, 38, 11, 108-114.

[29]. Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution Programs. Department of Computer Science. University of North Carolina.

[30]. Michalewicz, Z. y Janikow, C. Z. (1998) Handling Constraints in Genetic Algorithms. Department of Computer Science. University of North Carolina.

[31]. Minsky, M. (1985) The Society of Mind, New York: Simon and Schuster.

[32]. Mitchell, M. (1991) Review of L.D. Davis, Handbook of Genetic Algorithms. Santa Fe Institute.

[33]. Nwana, H. S. (1996) Software Agents: An Overview. Knowledge Engineering Review.

[34]. Ono, I. y Kobayashi, S. (1997) A Real Coded Genetic Algorithm for Function Optimization Using Unimodal Normal Distributed Crossover. Morgan Kaufmann.

[35]. Radcliffe, N.J. (1991) Formal analysis and random respectful recombination. In ICGA'91, pages 222-229.

[36]. Rand, W. y Stonedahl, F. (2007) The El Farol Bar Problem and Computational Effort: Why People Fail to use Bars Efficiently. Northwestern University, Evanston, IL.

[37]. Rechenberg, I. (1970) Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologisched Evolution. PhD thesis, Frommann-Holzboog, Stuttgart-Bad Cannstatt.

[38]. Reeves, C. (1993) Modern Heuristic Techniques for Combinatorial Problems. Blackwell Scientific Publications.

[39]. Renders, J.-M. y Bersini, H. (1994) Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. IEEE World Congress on Computational Intelligence.

[40]. Russell, S. J. y Norvig, P. (1995) Artificial Intelligence A Modern Approach. Prentice Hall.

[41]. Sánchez, S. (2012) Optimización Estructural y Topológica de Estructuras Morfológicamente no Definidas mediante Algoritmos Genéticos. Departamento de Ingeniería Mecánica y de Materiales. Tesis Doctoral, Universitat Politècnica de València.

[42]. Shoham, Y. (1990) Ageng-Oriented Programming. Inf. Téc. STAN-CS-90-1335, Computer Science Department, Stanford University.

[43]. Syswerda, G. (1989) Uniform crossover in genetic algorithms. In Proceedings of the 3rd International Conference on Genetic Algorithms, San Francisco, CA, USA.

[44]. Thierens, D. y Goldberg, D.E. (1994) Convergence models of genetic algorithm selection schemes. In Proceedings of the International Conference on Evolutionary Computation, London.

[45]. Wetzel, A. (1983) Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization. PhD thesis, University of Pittsburgh.

[46]. Wooldridge, M. (1997) Agent-Based Software Engineering. IEEE Proceedings Software Engineering, 144(1).

[47]. Wooldridge, M. (2000) Reasoning about Rational Agents. The MIT Press, Cambridge, Massachusetts.

[48]. Wooldridge, M. y Jennings, N. R. (1995) Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, 10(2).

Apéndice A

Líneas de código de agentes

```
/*****************************************************************
Agent that controls the play: AgTable.java
 *****************************************************************/

package examples.playKinito;

import jade.core.Agent;
import jade.core.AID;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.util.List;
import java.util.ArrayList;
import java.util.Hashtable;

/**
 * This agent has the following functionality:
 * <ul>
 * <li> It registers itself in the DF as TABLE
 * <li> Pick a random number between 1 and 6
 * <li> Chosse the next player and send the number as a message
 * <li> It waits for the player answer
 * <li> ...<missing>
 * </ul>
 * @author Nelly Salazar, MUSS - UPM
 * @version $Date: 2013/05/31 16:30:18 $ $Revision: 1.0 $
 **/

public class AgTable extends Agent {
    public static int nPlayers = 3;
    public final static String TABLE = "Table";
    public static boolean inPlay = false;
    public static int turn = 0;
    Hashtable<AID, Integer> coinsByPlayer = new Hashtable<AID, Integer>();
    public List<AID> players =  new ArrayList<AID>();
    public static boolean waitingResponse = false;
    public AID playerInit;
    public AID playerEnd;
    public static int nMax = 300;

    // <editor-fold defaultstate="collapsed" desc="Ontology">
```

```
    private class MatchPLAYOntology implements MessageTemplate.MatchExpression
{
        public boolean match(ACLMessage msg) {
            String content = msg.getContent();
            return (content != null && content.contains("PLAY"));
        }
    }
    private MessageTemplate templatePlay = MessageTemplate.and(
            MessageTemplate.MatchPerformative(ACLMessage.PROPOSE),
            new MessageTemplate(new MatchPLAYOntology()));

    private class MatchCOINSOntology implements
MessageTemplate.MatchExpression {
        public boolean match(ACLMessage msg) {
            String content = msg.getContent();
            return (content != null && content.contains("COINS"));
        }
    }
    private MessageTemplate templateCoins = MessageTemplate.and(
            MessageTemplate.MatchPerformative(ACLMessage.INFORM),
            new MessageTemplate(new MatchCOINSOntology()));

    private class MatchENDOntology implements MessageTemplate.MatchExpression
{
        public boolean match(ACLMessage msg) {
            String content = msg.getContent();
            return (content != null && content.contains("END"));
        }
    }
    private MessageTemplate templateEnd = MessageTemplate.and(
            MessageTemplate.MatchPerformative(ACLMessage.INFORM),
            new MessageTemplate(new MatchENDOntology()));
    // </editor-fold>

    protected void setup()
    {
        System.out.println("[Comment]" + getLocalName()+": se ha registrado en
el sistema.");

        // Creates its own description
        try
        {
           // Crea su descriptor
            DFAgentDescription dfd = new DFAgentDescription();
            ServiceDescription sd = new ServiceDescription();
            sd.setName(this.getName());
            sd.setType(TABLE);
            dfd.addServices(sd);
            // Registra su descripción en el DF
            DFService.register(this, dfd);
            dfd = null;
            sd = null;
        }
        catch (FIPAException e)
        {
            e.printStackTrace();
        }

        // <editor-fold defaultstate="collapsed" desc="BEHAV2 - Procesar
solicitud">
        addBehaviour(new CyclicBehaviour(this)
        {
            int i, j;
            public void action()
            {
                ACLMessage msg = myAgent.receive(templatePlay);
                if (msg != null)
                {
```

```
                    if (inPlay)
                    {
                        ACLMessage reply = msg.createReply();
                        reply.setPerformative(ACLMessage.REJECT_PROPOSAL);
                        reply.setContent("KO");
                        send(reply);
                        System.out.println("[Comment]" + getLocalName()+":
Rechaza la solicitud. Actualmente en juego.");
                    }
                    else
                    {
                        String content = msg.getContent();
                        content = content.replace("PLAY(", "");
                        content = content.replace(")", "");
                        int value = Integer.parseInt(content.replace("MONEY:",
""));
                        coinsByPlayer.put(msg.getSender(), value);
                        players.add(msg.getSender());

                        ACLMessage reply = msg.createReply();
                        reply.setPerformative(ACLMessage.ACCEPT_PROPOSAL);
                        reply.setContent("OK");
                        send(reply);
                        System.out.println("[Comment]" + getLocalName()+":
Acepta la solicitud.");

                        if (players.size() >= nPlayers)
                        {
                            inPlay = true;
                            java.util.Enumeration agPlayers =
coinsByPlayer.keys();
                            while(agPlayers.hasMoreElements() )
                            {
                                Object clave = agPlayers.nextElement();
                                Object valor = coinsByPlayer.get(clave);
                            }
                        }
                    }
                }
                else
                {
                    block();
                }
            }
        });
        // </editor-fold>

        // <editor-fold defaultstate="collapsed" desc="BEHAV3 - Iniciar
juego">
        addBehaviour(new CyclicBehaviour(this)
        {
            public void action()
            {
                if (inPlay)
                {
                    if (!waitingResponse)
                    {
                        AID player = players.get(turn);

                        System.out.println("Jugador seleccionado por Mesa: " +
player);

                        playerInit = player;
                        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                        msg.setContent("START");
                        msg.addReceiver(player);
                        send(msg);
                        waitingResponse = true;
```

```
                        nMax = nMax - 1;
                    }
                }
            }
        });
        // </editor-fold>

        // <editor-fold defaultstate="collapsed" desc="BEHAV8 - Actualizar
monedas">
         addBehaviour(new CyclicBehaviour(this)
        {
            int i, j;
            public void action()
            {
                try
                {
                    ACLMessage msg = myAgent.receive(templateCoins);
                    if (msg != null)
                    {
                        String content = msg.getContent();
                        content = content.replace("COINS(", "");
                        content = content.replace(")", "");
                        String[] arrayValues = content.split(",");
                        int value =
Integer.parseInt(arrayValues[0].replace("VALUE:", ""));
                        int end =
Integer.parseInt(arrayValues[1].replace("END:", ""));

                        coinsByPlayer.remove(msg.getSender());
                        coinsByPlayer.put(msg.getSender(), value);

                        if (end == 1)
                        {
                            java.util.Enumeration agPlayers =
coinsByPlayer.keys();
                            while(agPlayers.hasMoreElements() )
                            {
                               Object clave = agPlayers.nextElement();
                               int valor = coinsByPlayer.get(clave);
                               if (valor <= 0 || nMax < 1)
                               {
                                    inPlay = false;
                                    System.out.println("Fin del juego");
                               }
                            }
                            if (inPlay)
                            {
                                waitingResponse = false;
                                turn = turn + 1;
                                if (turn > (nPlayers - 1))
                                {
                                    turn = 0;
                                }
                            }
                            else
                            {
                                agPlayers = coinsByPlayer.keys();
                                while(agPlayers.hasMoreElements() )
                                {
                                    Object clave = agPlayers.nextElement();
                                    Object valor = coinsByPlayer.get(clave);
                                    System.out.println("[Comment]" + "Player:
"+clave.toString()+", Monedas:  " +valor.toString());
                                }
                            }
                        }

                    }
```

```
                    else
                    {
                        block();
                    }
                }
                catch (Exception ex)
                {
                    System.out.println("Error: " + ex.getMessage());
                }

            }
        });
        // </editor-fold>

    }
}
```

```
/*****************************************************************
Agent that controls the play: AgPlayer.java

 *****************************************************************/

package examples.playKinito;

import jade.core.Agent;
import jade.core.AID;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Hashtable;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Scanner;
import java.util.Calendar;
import java.io.*;

/**
 * This agent has the following functionality:
 * <ul>
 * <li> It registers itself in the DF as PLAYER
 * <li> Chosse the next player to play
 * <li> It waits for the player answer
 * </ul>
 * @author Nelly Salazar, MUSS - UPM
 * @version $Date: 2013/05/31 16:30:18 $ $Revision: 1.0 $
 **/

public class AgPlayer extends Agent {
    static Scanner sc = new Scanner(System.in);

    public static int nPlayers = 3;
    public static int nBits = 9;
    public static int applyNervousNumber = 1;
    public static int applyNervousHistory = 0;
    public static int useP3 = 1;
    public final static String PLAYER = "Player";
    public final static String TABLE = "Table";
    public static AID table;
    public static int ActualNumber = 0;
    public static int NewNumber = 0;
```

```
    public static int ActualResponse = 0;
    public static double ActualNervous = 0;
    public static int ActualResponseAg2 = 0;
    public static AID selectedPlayer;
    public static int[] personality1ByPlayer = {4, 3, 4};
    public static int[] personality2ByPlayer = {4, 0, 0};
    public static int[] personality3ByPlayer = {3, 2, 0};
    public static int[] inferenceMethod = {1, 2, 3};
    public static List<Integer> ownPersonality =  new ArrayList<Integer>();
    public static List<AID> players =  new ArrayList<AID>();
    public static double degP1 = 0.5;
    public static double degP2 = 0.5;
    public static List<String> TableAGHistoryProm = new ArrayList<String>();
    public static int nPlay = 0;

    // <editor-fold defaultstate="collapsed" desc="Ontology">
    private class MatchSTARTOntology implements
MessageTemplate.MatchExpression {
        public boolean match(ACLMessage msg) {
            String content = msg.getContent();
            return (content != null && content.contains("START"));
        }
    }
    private MessageTemplate templateStart = MessageTemplate.and(
            MessageTemplate.MatchPerformative(ACLMessage.INFORM),
            new MessageTemplate(new MatchSTARTOntology()));

    private class MatchINITIALOntology implements
MessageTemplate.MatchExpression {
        public boolean match(ACLMessage msg) {
            String content = msg.getContent();
            return (content != null && content.contains("INITIAL"));
        }
    }
    private MessageTemplate templateInitial = MessageTemplate.and(
            MessageTemplate.MatchPerformative(ACLMessage.INFORM),
            new MessageTemplate(new MatchINITIALOntology()));

    private class MatchRESPONSEOntology implements
MessageTemplate.MatchExpression {
        public boolean match(ACLMessage msg) {
            String content = msg.getContent();
            return (content != null && content.contains("RESPONSE"));
        }
    }
    private MessageTemplate templateResponse = MessageTemplate.and(
            MessageTemplate.MatchPerformative(ACLMessage.INFORM),
            new MessageTemplate(new MatchRESPONSEOntology()));

    private class MatchWINNEROntology implements
MessageTemplate.MatchExpression {
        public boolean match(ACLMessage msg) {
            String content = msg.getContent();
            return (content != null && content.contains("WINNER"));
        }
    }
    private MessageTemplate templateWinner = MessageTemplate.and(
            MessageTemplate.MatchPerformative(ACLMessage.INFORM),
            new MessageTemplate(new MatchWINNEROntology()));
    // </editor-fold>

    protected void setup()
    {
        Register(this);
        players.add(this.getAID());

        // <editor-fold defaultstate="collapsed" desc="BEHAV1 - Solicitar
jugar">
```

```
        addBehaviour(new OneShotBehaviour(this) {
            public void action() {
                table = GetTable(myAgent);
                if (table != null)
                {
                    int actualPlayerID =
GetAgentID(this.getAgent().getAID().getName());
                    ACLMessage msg = new ACLMessage(ACLMessage.PROPOSE);
                    msg.setContent("PLAY(MONEY:" + GetCoins(actualPlayerID) +
")");
                    msg.addReceiver(table);
                    send(msg);
                    System.out.println("[Comment]" + getLocalName() + ":
Solicita jugar");
                }
                else
                {
                    System.out.println("[Error]No se ha encontrado al agente
Table.");
                }
            }
        });
        // </editor-fold>

        // <editor-fold defaultstate="collapsed" desc="BEHAV4 - Iniciar
partida">
        addBehaviour(new CyclicBehaviour(this)
        {
            int i, j;
            public void action()
            {
                ACLMessage msg = myAgent.receive(templateStart);
                if (msg != null)
                {
                    int actualPlayerID =
GetAgentID(this.getAgent().getAID().getName());
                    int P1 = personality1ByPlayer[actualPlayerID];
                    int P2 = personality2ByPlayer[actualPlayerID];
                    int P3 = 0;
                    if (useP3 == 1)
                        P3 = personality3ByPlayer[actualPlayerID];

                    int value = (int) Math.floor((Math.random()*6+1));
                 ActualNumber = value;
                    ActualNervous = 0.0;
                    if (useP3 > 0)
                        ActualNervous = GetNervousnessByNumber(value, P3);

                    ActualResponse = 1;
                    if (value < 6)
                        ActualResponse = GetResponse(P1, P3, value);

                    if (ActualResponse == 0)
                    {
                        ActualNervous = GetNervousnessLie(P1, P2,
ActualNervous);
                        if (useP3 == 1)
                            NewNumber = GetNumber(value, P3);
                    }
                    else
                    {
                        ActualNervous = GetNervousnessNoLie(P1, P2,
ActualNervous);
                        if (useP3 == 1)
                            NewNumber = value;
                    }
                    int selectedPlayerID = GetTurn(actualPlayerID);
                    selectedPlayer = players.get(selectedPlayerID);
```

```
                    SetTurn(actualPlayerID, selectedPlayerID);

                    ACLMessage msgInitialData = new
ACLMessage(ACLMessage.INFORM);
                    msgInitialData.setContent("INITIAL(NUMBER:" + NewNumber +
",NERVOUSNESS:" + ActualNervous + ")");
                    msgInitialData.addReceiver(selectedPlayer);
                    send(msgInitialData);
                    System.out.println("[Comment]" + "Envía N: " + NewNumber +
" y nerviosismo " + ActualNervous);
                    nPlay = nPlay + 1;
                }
                else
                {
                    block();
                }
            }
        });
        // </editor-fold>

        // <editor-fold defaultstate="collapsed" desc="BEHAV5 - Responder
partida">
        addBehaviour(new CyclicBehaviour(this)
        {
            int i, j;
            public void action()
            {
                try
                {
                    ACLMessage msg = myAgent.receive(templateInitial);
                    if (msg != null)
                    {
                        String content = msg.getContent();
                        content = content.replace("INITIAL(", "");
                        content = content.replace(")", "");
                        String[] arrayValues = content.split(",");
                        int value =
Integer.parseInt(arrayValues[0].replace("NUMBER:", ""));
                        double nervous =
Double.parseDouble(arrayValues[1].replace("NERVOUSNESS:", ""));

                        int actualPlayerNumber =
GetAgentID(this.getAgent().getAID().getName());
                        int senderPlayerNumber =
GetAgentID(msg.getSender().getName());
                        int methodID = inferenceMethod[actualPlayerNumber];
                        int response = 0;
                        if (methodID == 1)
                            response =
ApplyGeneticAlgorithm(actualPlayerNumber, senderPlayerNumber, nervous, value);
                        else if (methodID == 2)
                            response = ApplyHistory(actualPlayerNumber,
senderPlayerNumber, value);
                        else{
                            response = (int) Math.floor((Math.random()*100));
                            if (response < 50)
                                response = 0;
                            else
                                response = 1;
                        }

                        ActualResponseAg2 = response;
                        ACLMessage reply = msg.createReply();
                        reply.setPerformative(ACLMessage.INFORM);
                        reply.setContent("RESPONSE(VALUE:" + response + ")");
                        send(reply);
                        System.out.println("[Comment]" + "Envía la respuesta "
+ response);
```

```
                    }
                    else
                    {
                        block();
                    }
                }
                catch (Exception ex)
                {
                    System.out.println("[Error]Response: " + ex.getMessage());
                }
            }
        });
        // </editor-fold>

        // <editor-fold defaultstate="collapsed" desc="BEHAV6 - Establecer
ganador">
        addBehaviour(new CyclicBehaviour(this)
        {
            int i, j;
            public void action()
            {
                try
                {
                    ACLMessage msg = myAgent.receive(templateResponse);
                    if (msg != null)
                    {
                        System.out.println(msg.getContent());
                        String content = msg.getContent();
                        content = content.replace("RESPONSE(", "");
                        content = content.replace(")", "");
                        String[] arrayValues = content.split(",");
                        int value =
Integer.parseInt(arrayValues[0].replace("VALUE:", ""));

                        int actualPlayerNumber =
GetAgentID(this.getAgent().getAID().getName());
                        int response = 1;
                        int coins = GetCoins(actualPlayerNumber)+
ActualNumber;
                        if (ActualResponse == value)
                        {
                            response = 0;
                            coins = GetCoins(actualPlayerNumber)-
ActualNumber;
                        }
                        SetCoins(actualPlayerNumber, coins);
                        SetHistoryTable("Selector", actualPlayerNumber,
GetInitialNervous(actualPlayerNumber), ActualNervous, ActualNumber,
ActualResponse, value, response);
                        if (applyNervousHistory > 0)
                            SetInitialNervous(actualPlayerNumber,
GetNervousnessByHistory(actualPlayerNumber));

                        ACLMessage msgAnswer = new
ACLMessage(ACLMessage.INFORM);
                        msgAnswer.setContent("COINS(VALUE:" +
GetCoins(actualPlayerNumber) + ",END:0)");
                        msgAnswer.addReceiver(table);
                        send(msgAnswer);

                        if (response == 0)
                            response = 1;
                        else
                            response = 0;
                        ACLMessage reply = msg.createReply();
                        reply.setPerformative(ACLMessage.INFORM);
                        reply.setContent("WINNER(VALUE:" + response +
",MONEY:" + ActualNumber + ")");
```

```
                        send(reply);
                        System.out.println("[Comment]" + "Envía la respuesta
al agente j.");
                    }
                    else
                    {
                        block();
                    }
                }
                catch (Exception ex)
                {
                    System.out.println("[Error]Establecer Ganador: " +
ex.getMessage());
                }

            }
        });
        // </editor-fold>

        // <editor-fold defaultstate="collapsed" desc="BEHAV7 - Finalizar
partida">
        addBehaviour(new CyclicBehaviour(this)
        {
            int i, j;
            public void action()
            {
                try
                {
                    ACLMessage msg = myAgent.receive(templateWinner);
                    if (msg != null)
                    {
                        String content = msg.getContent();
                        content = content.replace("WINNER(", "");
                        content = content.replace(")", "");
                        String[] arrayValues = content.split(",");
                        int value =
Integer.parseInt(arrayValues[0].replace("VALUE:", ""));
                        int money =
Integer.parseInt(arrayValues[1].replace("MONEY:", ""));

                        int actualPlayerNumber =
GetAgentID(this.getAgent().getAID().getName());
                        int senderPlayerNumber =
GetAgentID(msg.getSender().getName());
                        int coins = GetCoins(actualPlayerNumber)+ money;

                        if (value == 0)
                            coins = GetCoins(actualPlayerNumber)-money;
                        SetCoins(actualPlayerNumber, coins);
                        SetCorrectResponse(actualPlayerNumber,
senderPlayerNumber, value);
                        SetHistoryTable("Selected", actualPlayerNumber,
InitialNervousAg2, ActualNervousAg2, ActualNumber, ActualResponse,
ActualResponseAg2, value);
                        PastInfo.add(nPlay + "_" + ActualNumber + "_" +
ActualResponseAg2 + "_" + ActualResponse);
                        if (applyNervousHistory > 0)
                            SetInitialNervous(actualPlayerNumber,
GetNervousnessByHistory(actualPlayerNumber));

                        ACLMessage msgAnswer = new
ACLMessage(ACLMessage.INFORM);
                        msgAnswer.setContent("COINS(VALUE:" +
GetCoins(actualPlayerNumber) + ",END:1)");
                        msgAnswer.addReceiver(table);
                        send(msgAnswer);
                        System.out.println("[Comment]" + "Envía final a
mesa.");
```

```
                    }
                    else
                    {
                        block();
                    }
                }
                catch (Exception ex)
                {
                    System.out.println("[Error]End: " + ex.getMessage());
                }

            }
        });
        // </editor-fold>
    }
```

Apéndice B

Líneas de código para hallar la emoción

```
// <editor-fold defaultstate="collapsed" desc="Personality">
    public static int GetNumber(int number, int p3)
    {
        double previous = 0;
        int num = 0;
        previous = number*1.0 + p3*1.2;

        if (previous < 1.5)
            num = 1;
        else if (previous < 2.5)
            num = 2;
        else if (previous < 3.5)
            num = 3;
        else if (previous < 4.5)
            num = 4;
        else if (previous < 5.5)
            num = 5;
        else
            num = 6;

        return num;
    }

    public static Double GetNervousnessByNumber(int number, int p3){
        SetLog("Nerviosismo actual = " + ActualNervous);
        double nervous = 0.0;
        double factor = 0.15;
        double p3Value = (p3*1.0)/(4*1.0);
        if (p3Value == 0)
            p3Value = 0.01;
        if (p3Value == 1)
            p3Value = 0.99;

        nervous = ((6.0 - number*1.0)/5.0)*(p3Value*1.0);

        SetLog("Nerviosismo por número = " + nervous);
        if (nervous < 0.2)
            nervous = 0;
        else if (nervous < 0.4)
            nervous = 1;
        else if (nervous < 0.6)
            nervous = 2;
        else if (nervous < 0.8)
            nervous = 3;
        else
```

```
            nervous = 4;

        return nervous;
    }

    public static Integer GetResponse(int P1, int P3, int number){
        int Response = 1;
        double Max = 0.0;

        if (useP3 > 1){
            double p1Value = (P1*1.0)/(4*1.0);
            double p3Value = (P3*1.0)/(4*1.0);
            if (p1Value == 0)
                p1Value = 0.01;
            if (p1Value == 1)
                p1Value = 0.99;
            if (p3Value == 0)
                p3Value = 0.01;
            if (p3Value == 1)
                p3Value = 0.99;
            Max = (p1Value*0.8 + (((6.0 -
number*1.0)/5.0*p3Value*1.0))*0.2)*100.0;
        }
        else
        {
            switch (P1) {
            case 0:  Max = 10.0;
                     break;
            case 1:  Max = 30.0;
                     break;
            case 2:  Max = 50.0;
                     break;
            case 3:  Max = 70.0;
                     break;
            case 4:  Max = 90.0;
                     break;
            default: Max = 0.0;
                     break;
        }
        }

        SetLog("Valor máximo = " + Max);
        int value = (int) Math.floor((Math.random()*100+1));
        SetLog("Valor obtenido = " + value);
        if (value <= Max)
            Response = 0;

        return Response;
    }

    public static Integer GetResponse(String p1, String p3, int number){
        int response = GetResponse(binToDec(p1), binToDec(p3), number);

        return response;
    }

    public static double GetNervousnessLie(int P1, int P2, double nervous){
        double Nervousness = 0;
        SetLog("Nerviosismo actual = " + nervous);
        nervous = nervous/(4.0);

        if (useP3 >1)
        {
            if (nervous == 0)
            nervous = 0.01;
            if (nervous == 1)
                nervous = 0.99;
            double p1Value = (P1*1.0)/(4*1.0);
```

```
            if (p1Value == 0)
                p1Value = 0.01;
            if (p1Value == 1)
                p1Value = 0.99;
            double p2Value = (P2*1.0)/(4*1.0);
            if (p2Value == 0)
                p2Value = 0.01;
            if (p2Value == 1)
                p2Value = 0.99;
            Nervousness = nervous - (p1Value*1.0)*degP1;
            if (Nervousness < 0.125)
                Nervousness = 0.0;
            else if (Nervousness < 0.375)
                Nervousness = 1.0;
            else if (Nervousness < 0.625)
                Nervousness = 2.0;
            else if (Nervousness < 0.875)
                Nervousness = 3.0;
            else
                Nervousness = 4.0;
            Nervousness = Nervousness/4.0;
            if (Nervousness == 0)
                Nervousness = 0.01;
            if (Nervousness == 1)
                Nervousness = 0.99;
            SetLog("Nerviosismo después de P1 = " + Nervousness);
            if (Nervousness >= 0.5)
                Nervousness = Nervousness - (p2Value*1.0)*degP2;
            else
                Nervousness = Nervousness + (p2Value*1.0)*degP2;
            SetLog("Nerviosismo después de P2 = " + Nervousness);
        }
        else
        {
            Nervousness = nervous - (P1*1.0)/(4.0)*degP1;
        }

        if (Nervousness < 0.125)
            Nervousness = 0;
        else if (Nervousness < 0.375)
            Nervousness = 1;
        else if (Nervousness < 0.625)
            Nervousness = 2;
        else if (Nervousness < 0.875)
            Nervousness = 3;
        else
            Nervousness = 4;
        return Nervousness;
    }

    public static double GetNervousnessNoLie(int P1, int P2, double nervous){
        double Nervousness = 0;
        SetLog("Nerviosismo actual = " + nervous);
        nervous = nervous/(4*1.0);

        if (useP3 >1)
        {
            if (nervous == 0)
                nervous = 0.01;
            if (nervous == 1)
                nervous = 0.99;
            double p2Value = (P2*1.0)/(4*1.0);
            if (p2Value == 0)
                p2Value = 0.01;
            if (p2Value == 1)
                p2Value = 0.99;

            if (nervous >= 0.5)
```

```
                Nervousness = nervous - (p2Value*1.0)*degP2;
            else
                Nervousness = nervous + (p2Value*1.0)*degP2;
            SetLog("Nerviosismo después de P2 = " + Nervousness);
        }
        else
        {
            if (nervous >= 0.5)
                Nervousness = nervous - (P2*1.0)/4.0*degP2;
            else
                Nervousness = nervous + (P2*1.0)/4.0*degP2;
            SetLog("Nerviosismo después de P2 = " + Nervousness);
        }

        if (Nervousness < 0.125)
            Nervousness = 0;
        else if (Nervousness < 0.375)
            Nervousness = 1;
        else if (Nervousness < 0.625)
            Nervousness = 2;
        else if (Nervousness < 0.875)
            Nervousness = 3;
        else
            Nervousness = 4;
        return Nervousness;
    }
    // </editor-fold>
```

Apéndice C

Líneas de código del algoritmo genético

```
// <editor-fold defaultstate="collapsed" desc="Genetics">
    public static Integer ApplyGeneticAlgorithm(int actualPlayerNumber, int
senderPlayerNumber, double nervous, int value){
        int response = 0;
        String[][] fitnessTable = new String[nPopulation][5];
        List<String> population = new ArrayList<String>();

        fitnessTable = GetFitnessTable(actualPlayerNumber,
senderPlayerNumber);

        if (fitnessTable[0][0] == null)
        {
            SetLog("Genera la población inicial aleatoria");
            population = GetRandomGeneration();
        }
        else
        {
            SetLog("Genera la población evolucionando");
            population = ApplyGeneticOperators(fitnessTable);
        }
        SetLog("Calcula el fitness de cada individuo");
        fitnessTable = GetFitness(population, nervous, value);
        SetFitnessTable(actualPlayerNumber, senderPlayerNumber, fitnessTable);

        //Obtener la respuesta
        response = Integer.parseInt(fitnessTable[0][3]);
        SetLog("Respuesta: " + response);
        return response;
    }

    public static List<String> GetRandomGeneration() {
        List<String> population =  new ArrayList<String>();
        int i = 1;
        while (i <= nPopulation)
        {
            try
            {
                String single = "";
                int valueP1 = (int) Math.floor((Math.random()*100));
                if (valueP1 < 20)
                    valueP1 = 0;
                else if (valueP1 < 40)
                    valueP1 = 1;
                else if (valueP1 < 60)
                    valueP1 = 2;
```

```
                else if (valueP1 < 80)
                    valueP1 = 3;
                else
                    valueP1 = 4;

                int valueP2 = (int) Math.floor((Math.random()*100));
                if (valueP2 < 20)
                    valueP2 = 0;
                else if (valueP2 < 40)
                    valueP2 = 1;
                else if (valueP2 < 60)
                    valueP2 = 2;
                else if (valueP2 < 80)
                    valueP2 = 3;
                else
                    valueP2 = 4;

                int valueE1 = (int) Math.floor((Math.random()*100));
                if (valueE1 < 20)
                    valueE1 = 0;
                else if (valueE1 < 40)
                    valueE1 = 1;
                else if (valueE1 < 60)
                    valueE1 = 2;
                else if (valueE1 < 80)
                    valueE1 = 3;
                else
                    valueE1 = 4;

                int valueP3 = (int) Math.floor((Math.random()*100));
                if (valueP3 < 20)
                    valueP3 = 0;
                else if (valueP3 < 40)
                    valueP3 = 1;
                else if (valueP3 < 60)
                    valueP3 = 2;
                else if (valueP3 < 80)
                    valueP3 = 3;
                else
                    valueP3 = 4;
                single = padLeft(Integer.toBinaryString(valueP1), 3, "0")
                        + padLeft(Integer.toBinaryString(valueP2), 3, "0");
                if (nBits > 6)
                   single = single + padLeft(Integer.toBinaryString(valueP3),
3, "0");
                population.add(single);
                SetLog("Individuo " + i + ": " + single);
                i = i + 1;
            }
            catch(RuntimeException  e)
            {
                System.out.println("Error getRandomGeneration: " +
e.getMessage());
            }
        }

        return population;
    }

    public static List<String> ApplyGeneticOperators(String[][] fitnessTable)
{
        int pElitism = 20;
        int pCrossover = 70;
        int pItemMutation = 10;
        int pMutation = 20;

        List<String> population =  new ArrayList<String>();
        List<String> populationSelect =  new ArrayList<String>();
```

```
        List<String> populationCrossover =  new ArrayList<String>();
        List<String> populationMutation =  new ArrayList<String>();
        int point = 0;
        int father1 = 0;
        int father2 = 0;
        String father1String = "";
        String father2String = "";
        String newChild = "";

        SetLog("MUESTRA TABLA FITNESS ANTERIOR");
        for (int m = 0; m < fitnessTable.length; m++)
        {
            SetLog(fitnessTable[m][0] + " " + fitnessTable[m][1] + " " +
fitnessTable[m][2] + " " + fitnessTable[m][3] + " " + fitnessTable[m][4]);
        }

        //Obtiene la respuesta correcta
        int correctAnswer = Integer.parseInt(fitnessTable[0][4]);
        SetLog("Respuesta correcta: " + correctAnswer);
        int response = Integer.parseInt(fitnessTable[0][3]);
        SetLog("Respuesta dada: " + response);
        int column = correctAnswer + 1;

        SetLog("Columna: " + column);
        int i = 0;
        //Selección
        while (i < nPopulation)
        {
            //Selecciona dos individuos al azar
            father1 = (int) Math.floor((Math.random()*(nPopulation - 1)));
            father2 = (int) Math.floor((Math.random()*(nPopulation - 1)));
            double fitnessFather1 =
Double.parseDouble(fitnessTable[father1][column]);
            double fitnessFather2 =
Double.parseDouble(fitnessTable[father2][column]);
            String fatherToAdd = "";

            //Obtiene la respuesta correcta
            if (fitnessFather1 < fitnessFather2)
                fatherToAdd = fitnessTable[father1][0];
            else if (fitnessFather1 > fitnessFather2)
                fatherToAdd = fitnessTable[father2][0];
            else
            {
                int selected = (int) Math.floor((Math.random()*1));
                if (selected == 0)
                    fatherToAdd = fitnessTable[father1][0];
                else
                    fatherToAdd = fitnessTable[father2][0];
            }
            SetLog("Selected: " + fatherToAdd);
            populationSelect.add(fatherToAdd);
            i = i + 1;
        }

        //Cruce
        i = 0;
        while(i < nPopulation)
        {
            father1 = (int) Math.floor((Math.random()*(nPopulation)));
            father2 = (int) Math.floor((Math.random()*(nPopulation)));
            father1String = populationSelect.get(father1);
            father2String = populationSelect.get(father2);

            SetLog("father1String: " + father1String);
            SetLog("father2String: " + father2String);

            int crossover = (int) Math.floor((Math.random()*100));
```

```
            SetLog("Hay cruce: " + crossover);
            if (crossover <= pCrossover)
            {
                point = (int) Math.floor((Math.random()*(nBits)));
                SetLog("Punto de Cruce: " + point);
                newChild = father1String.substring(0, point + 1) +
father2String.substring(point + 1);
                SetLog("Hijo 1: " + newChild);
                if (binToDec(newChild.substring(0, 3)) > 4)
                    newChild = "100" + newChild.substring(3);
                if (binToDec(newChild.substring(3, 6)) > 4)
                    newChild = newChild.substring(0, 3) + "100" +
newChild.substring(6);
                if (binToDec(newChild.substring(6)) > 4)
                    newChild = newChild.substring(0, 6) + "100";
                populationCrossover.add(newChild);
                newChild = father2String.substring(0, point + 1) +
father1String.substring(point + 1);
                SetLog("Hijo 2: " + newChild);
                if (binToDec(newChild.substring(0, 3)) > 4)
                    newChild = "100" + newChild.substring(3);
                if (binToDec(newChild.substring(3, 6)) > 4)
                    newChild = newChild.substring(0, 3) + "100" +
newChild.substring(6);
                if (binToDec(newChild.substring(6)) > 4)
                    newChild = newChild.substring(0, 6) + "100";
                populationCrossover.add(newChild);
            }
            else
            {
                populationCrossover.add(father1String);
                populationCrossover.add(father2String);
            }

            i = i + 2;
        }

        //Mutación
        i = 0;
        while (i < nPopulation)
        {
            String item = populationSelect.get(i);
            boolean muted = false;

            for (int k = 0; k < item.length(); k++)
            {
                int mutation = (int) Math.floor((Math.random()*100));
                if (mutation <= pItemMutation)
                {
                    int value = Integer.parseInt(item.substring(k));
                    if (value == 0)
                        value = 1;
                    else
                        value = 0;
                    item = item.substring(0, k) + value + item.substring(k
+1);

                    muted = true;
                }
            }

            if (muted)
            {
                SetLog("Mutación: " + item);
                if (binToDec(item.substring(0, 3)) > 4)
                    item = "100" + item.substring(3);
                if (binToDec(item.substring(3, 6)) > 4)
                    item = item.substring(0, 3) + "100" + item.substring(6);
                if (binToDec(item.substring(6)) > 4)
```

```
                    item = item.substring(0, 6) + "100";
                populationMutation.add(item);
            }

            i = i + 1;
        }

        //Reemplazo
        //Ordena la matriz por el valor de fitness
        if (column == 1)
            Arrays.sort(fitnessTable, new Comparator<String[]>() {
                @Override
                public int compare(final String[] entry1, final String[]
entry2) {
                    final String p1 = entry1[1];
                    final String p2 = entry2[1];
                    return p1.compareTo(p2);
                }
            });
        else
            Arrays.sort(fitnessTable, new Comparator<String[]>() {
                @Override
                public int compare(final String[] entry1, final String[]
entry2) {
                    final String p1 = entry1[2];
                    final String p2 = entry2[2];
                    return p1.compareTo(p2);
                }
            });

        i = 0;
        int j = 0;
        while (i < nPopulation)
        {
            while (i < (int)pElitism*nPopulation/100)
            {
                SetLog("Agente Elite " + i + ": " + fitnessTable[i][0]);
                population.add(fitnessTable[i][0]);
                i = i + 1;
            }

            while (j < (int)pMutation*nPopulation/100)
            {
                try
                {
                    SetLog("Agente Mutación " + i + ": " +
populationMutation.get(i));
                    population.add(populationMutation.get(j));
                    i = i + 1;
                    j = j + 1;
                }
                catch(Exception e)
                {
                    j = j + 1;
                }
            }

            try
            {
                SetLog("Agente Cruce " + i + ": " +
populationCrossover.get(i));
                population.add(populationCrossover.get(i));
                i = i + 1;
            }
            catch(Exception e){}
        }

        return population;
```

```
    }

    public static String[][] GetFitness(List<String> initialPopulation, double
nervous, int value) {
        //List<String> population =  new ArrayList<String>();
        //Hashtable<String, Integer> responseTable = new Hashtable<String,
Integer>();
        String[][] fitnessTable = new String[nPopulation][5];
        int j = 0;
        double sumLie = 0;
        double sumNoLie = 0;

        try
        {
            //1° Encuentra el valor de adecuación de cada individuo
            for(Iterator<String> i = initialPopulation.iterator();
i.hasNext(); ) {
                String item = i.next();
                String p1String = item.substring(0, 3);
                String p2String = item.substring(3, 6);
                String p3String = "";
                String m1String = "";
                if (nBits > 6)
                    m1String = item.substring(6);
                if (useP3 == 1)
                    p3String = item.substring(6);
                int p1 = binToDec(p1String);
                int p2 = binToDec(p2String);
                int p3 = 0;
                double m1Lie = 0;
                double m1NoLie = 0;
                if (nBits > 6)
                {
                    m1Lie = binToDec(m1String)*1.0;
                    m1NoLie = binToDec(m1String)*1.0;
                }
                if (useP3 == 1)
                {
                    p3 = binToDec(p3String);
                    //Calcular m1
                    double previous = value - p3*1.2;
                    int realNumber = 0;
                    if (previous < 1.5)
                        realNumber = 1;
                    else if (previous < 2.5)
                        realNumber = 2;
                    else if (previous < 3.5)
                        realNumber = 3;
                    else if (previous < 4.5)
                        realNumber = 4;
                    else if (previous < 5.5)
                        realNumber = 5;
                    else
                        realNumber = 6;

                    m1Lie = GetNervousnessByNumber(realNumber, p3);
                    m1NoLie = GetNervousnessByNumber(value, p3);
                }
                double nervousLie = GetNervousnessLie(p1, p2, m1Lie);
                double nervousNoLie = GetNervousnessNoLie(p1, p2, m1NoLie);
                SetLog("nervousLie:" + nervousLie);
                SetLog("nervousNoLie:" + nervousNoLie);
                double fitnessLie = Math.abs(nervous - nervousLie);
                double fitnessNoLie = Math.abs(nervous - nervousNoLie);

                fitnessTable[j][0] = item; // .put(item, fitness);
                fitnessTable[j][1] = Double.toString(fitnessLie);
                fitnessTable[j][2] = Double.toString(fitnessNoLie);
```

```
                sumLie = sumLie + fitnessLie;
                sumNoLie = sumNoLie + fitnessNoLie;
                j = j + 1;
            }
            //double promLie = sumLie/nPopulation;
            //double promNoLie = sumNoLie/nPopulation;
            //String promedios = promLie + ";" + promNoLie;

            SetLog("sumLie:" + sumLie);
            SetLog("sumNoLie:" + sumNoLie);
            int response = 0;
            if (sumLie < sumNoLie)
            {
                response = 0;
                //promedios = promedios + ";" +promLie;
            }
            if (sumLie > sumNoLie)
            {
                response = 1;
                //promedios = promedios + ";" +promNoLie;
            }
            if (sumLie == sumNoLie)
            {
                response = (int) Math.floor((Math.random()*1));
                //promedios = promedios + ";" +promLie;
            }
            //TableAGHistoryProm.add(promedios);
            SetLog("response:" + response);
            for(int i = 0; i < fitnessTable.length; i++)
            {
                fitnessTable[i][3] = Integer.toString(response);
            }

            //Ordena la matriz por el valor de fitness
            if (response == 0)
                Arrays.sort(fitnessTable, new Comparator<String[]>() {
                    @Override
                    public int compare(final String[] entry1, final String[]
entry2) {
                        final String p1 = entry1[1];
                        final String p2 = entry2[1];
                        return p1.compareTo(p2);
                    }
                });
            else
                Arrays.sort(fitnessTable, new Comparator<String[]>() {
                    @Override
                    public int compare(final String[] entry1, final String[]
entry2) {
                        final String p1 = entry1[2];
                        final String p2 = entry2[2];
                        return p1.compareTo(p2);
                    }
                });

            String bestItem = fitnessTable[0][0].substring(6);
            //InitialNervousAg2 = binToDec(bestItem);
            ActualNervousAg2 =
Double.parseDouble(fitnessTable[0][response+1]);

            for(int i = 0; i < fitnessTable.length; i++)
            {
                SetLog("fitnessTable:" + fitnessTable[i][0] + "; " +
fitnessTable[i][1] + "; "
                        + fitnessTable[i][2] + "; " + fitnessTable[i][3]);
            }
        }
        catch (Exception ex)
```

```
        {
            System.out.println("Error GetFitness: " + ex.getMessage());
        }

        return fitnessTable;
    }
    // </editor-fold>
```

Apéndice D

JADE

JADE (Java Agent DEvelopment Framework) es un entorno de desarrollo de software cuyo objetivo es simplificar la implementación y a la vez asegurar que éste se realice acorde con las especificaciones definidas por FIPA. JADE puede ser considerado como un agente middleware que implementa una Plataforma de Agente y un marco de desarrollo que gestiona todos los aspectos internos de un agente que no son dependientes de la aplicación como por ejemplo transporte de mensajes, codificación y decodificación o el ciclo de vida de un agente.

Fue concebido y desarrollado por Telecom Italia Lab y en el año 2000 se integra a la comunidad de código abierto bajo licencia LGPL. En 2003 forma, en conjunto con Motorola, la organización JADE Governing Board cuya misión es de promover la evolución y adopción de JADE en la industria de las telecomunicaciones móviles.

La plataforma se puede distribuir en varias máquinas (que no requieren el mismo sistema operativo) y la configuración puede ser controlada a través de una GUI remota (Ver Figura 31). JADE está completamente implementado en Java y requiere como mínimo la versión 1.4.

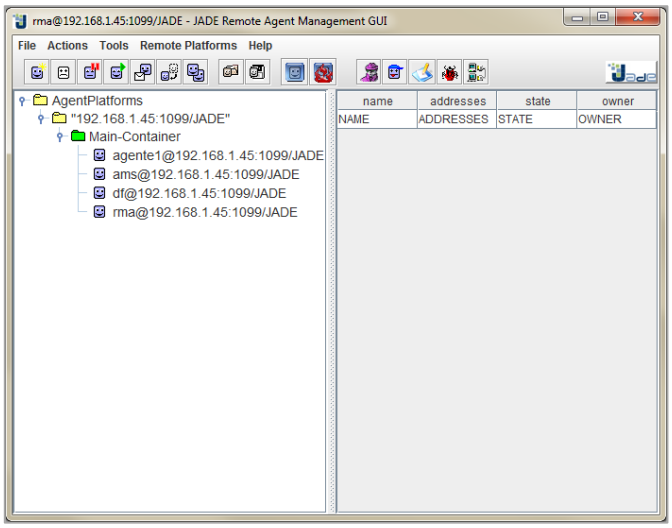

Figura 31 – Interface gráfica de JADE.

A continuación se exponen los conceptos principales asociados con JADE y se describe su funcionamiento.

1. Agentes

La programación de un agente JADE consiste en la definición de una clase Java que los representa en la que se implementarán los comportamientos que va a manifestar. Cada agente será una instancia de una clase que deber heredar de la clase *Agent*.

Las características que debe cumplir un agente JADE son:

- Tiene un nombre único en el entorno de ejecución.
- Se implementa como un único hilo de ejecución (*single-threaded*).
- Tiene un método de inicio (*setup*) y otro de fin (*takeDown*).
  - El método protegido *setup*() sirve para inicializar el agente incluyendo instrucciones que especificarán la ontología a utilizar y los comportamientos asociados al agente. Se invoca al comenzar la ejecución del agente.
  - El método protegido *takeDown*() sirve para liberar recursos antes de la eliminación del agente. Este método es invocado cuando se realiza una llamada al método *doDelete*(), que es el que realmente da por finalizada la ejecución del agente.
  - Ambos métodos deben ser sobrescritos.
- En su implementación se define una clase interna por cada uno de los comportamientos asociados al agente. Estos comportamientos se utilizan básicamente para el envío y recepción de mensajes, aunque también se pueden utilizar para realizar otras tareas.

Un agente está sujeto a un ciclo de vida en el que se definen los estados en los cuales se puede encontrar el agente, así como los cambios que se pueden realizar entre los diferentes estados. El ciclo de vida de un agente JADE sigue el ciclo propuesto por FIPA, es decir, cumple con la propuesta del estándar de interoperabilidad entre agentes más aceptado. Estos estados son:

- Iniciado: El objeto Agente está creado pero todavía no se ha registrado en el AMS, no tiene nombre ni dirección y tampoco se puede comunicar con otros agentes.
- Activo: El Agente está registrado en el AMS, tiene un nombre, una dirección y puede acceder a todas las opciones de JADE.
- Suspendido: El Agente está parado. Su hilo de ejecución está detenido y no ejecuta ningún Comportamiento.
- En espera: El Agente está bloqueado esperando por algo. Su hilo de ejecución está dormido en un monitor de java y se despertará cuando se cumpla una cierta condición (cuando reciba un mensaje).
- Desconocido: El Agente ha sido eliminado. El hilo de ejecución ha terminado y se ha eliminado del registro del AMS.
- Tránsito: Un Agente móvil entra en este estado mientras está migrando a una nueva localización. El sistema sigue guardando los mensajes en el buffer hasta que el agente vuelve a estar activo.

Un agente puede cambiar de un estado a otro a través de transiciones. Estas transiciones pueden ser ejecutadas a través de métodos disponibles en la clase *Agent* y ser capturados por métodos que se pueden sobrescribir.

Acción	Descripción	Método que realiza la acción	Método que captura la acción
Crear	Creación o instalación de un nuevo agente.	Constructor	setup()
Invocar	Invocación de un nuevo agente.		
Suspender	Pone un agente en estado suspendido. Puede ser iniciado por el agente o por el AMS.	doSuspend()	
Reanudar	Continúa con la ejecución de un agente que se encuentra en estado suspendido. Sólo puede ser iniciado por el AMS.		
Esperar	Pone un agente en estado de espera. Sólo puede ser iniciado por el agente.	doWait()	
Despertar	Continua con la ejecución de un agente que se encuentra en estado de espera. Sólo puede ser iniciado por el AMS.	doWake()	
Mover	Pone un agente en otro contenedor cambiando su estado al de tránsito. Sólo puede ser iniciado por el agente.	doMove()	beforeMove() y afterMove()
Ejecutar	Continúa con la ejecución de un agente que se encuentra en estado de tránsito. Sólo puede ser iniciado por el AMS.		
Destruir	La terminación normal o forzosa de un agente. Sólo puede ser iniciado por el AMS y no puede ser ignorado por el agente.	doDelete()	takeDown()

Tabla 13 – Métodos de la clase Agent.

Gráficamente se puede visualizar de la siguiente forma:

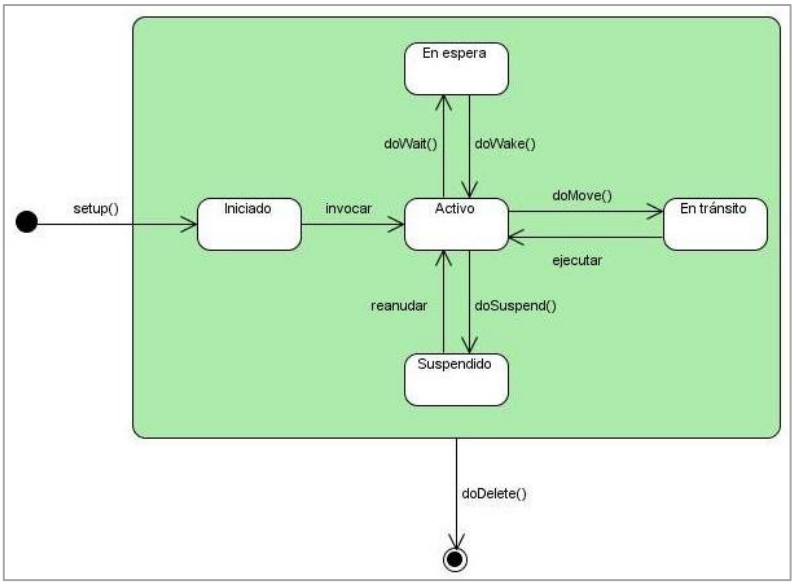

Figura 32 – Métodos de la clase Agent.

2. Comportamientos

Un comportamiento hace referencia a una funcionalidad que incorpora el agente y especifican tareas o servicios que realiza un agente para lograr sus objetivos. Cada tarea del agente será una instancia de una clase que deber heredar de la clase *Behaviour*.

Los agentes están programados en base a sus comportamientos. La programación basada en comportamientos debe realizar los siguientes pasos:

1. Determinar qué debe ser capaz de hacer el agente.
2. Asociar cada funcionalidad con un comportamiento.
3. Escoger el tipo de comportamientos
4. Dejar a JADE la tarea del *scheduling* (un solo comportamiento se está ejecutando en cada instante).

Toda clase que herede de *Behaviour* deberá implementar:

- el método *action*().
  - Este método define la acción a ser ejecutada cuando se ejecute el comportamiento. Debe incluir el código de las acciones a realizar cuando se ejecute el comportamiento.
  - Es invocado cuando se produce el evento asociado al comportamiento.
  - Es recomendable que los métodos *action*() no tengan un tiempo de ejecución alto ya que mientras que se ejecutan no pueden ser interrumpidos por otro comportamiento.
- el método *done*().
  - Es invocado cuando finaliza la ejecución del método *action*().
  - Este método determina si el comportamiento ha sido completado o no. Devuelve un booleano (true si ha terminado o false en caso contrario).
  - Si el comportamiento ha finalizado, éste se elimina de la cola de comportamientos activos.
  - Se puede utilizar una marca que se activa cuando se quiere que finalice el comportamiento (se evalúa su valor en el método *done*()).

Un comportamiento también puede ser bloqueado utilizando el método *block*(). Este método permite bloquear un comportamiento hasta que algún acontecimiento ocurra (típicamente, hasta que un mensaje llegue). Este no afecta a los demás comportamientos de un agente.

Cuando el método *action*() termina, el método *block*() coloca el comportamiento en la cola de comportamientos bloqueados. Además, un objeto de la clase *Behaviour* puede bloquearse durante una cantidad limitada de tiempo que se pasa por valor al método *block*(), expresado en milisegundos.

Un comportamiento bloqueado puede reiniciar su ejecución (desbloquearse) cuando alguna de las siguientes condiciones ocurre:

- El agente al que pertenece ese comportamiento recibe un mensaje ACL. En este caso el comportamiento se saca de la cola de comportamientos bloqueados y se coloca al

final de la cola de comportamientos activos. Cuando se produce la llegada de un mensaje, todos los objetos de la cola de bloqueados se planifican y deben comprobar si el mensaje es para ellos o no. en el caso de que un objeto no sea el destinatario debe volver a bloquearse.

- Una interrupción asociada con este comportamiento por el método block() expira. Por ejemplo, si un objeto se ha bloqueado durante dos segundos, al finalizar este tiempo, el objeto se desbloqueará.
- El método *restart*() es llamado explícitamente por el comportamiento, así se fuerza el desbloqueo.

Cada agente tiene un planificador o *scheduler* de comportamientos. La política de planificación se realiza de forma preemptiva para todos los comportamientos activos de la cola circular, es decir, ejecutando un comportamiento hasta que libera el control, sin poder interrumpirla hasta que ésta acabe (esto ocurre cuando finaliza el método *action*()).

El flujo de control de un agente básico corresponde con: Inicialización, realización de la tarea y limpieza y finalización; es decir, el camino que atraviesa un agente desde que comienza su ejecución hasta que finaliza y se elimina. Como puede verse, lo primero a ejecutar es el método *setup*(). Tras esto se comprueba que el agente sigue vivo y después se selecciona el siguiente comportamiento a ejecutar del conjunto de comportamientos que aún le quedan al agente. Se ejecuta su método b.*action*() y tras esto se pregunta si ha finalizado. Es posible que no lo haya hecho ya que un comportamiento puede ser o un simple trozo de código que se ejecuta una sola vez o bien varias veces dependiendo de otros factores. Si está ejecutado se elimina del conjunto de comportamientos del agente y no vuelve a ejecutarse. En otro caso, se vuelve a comenzar.

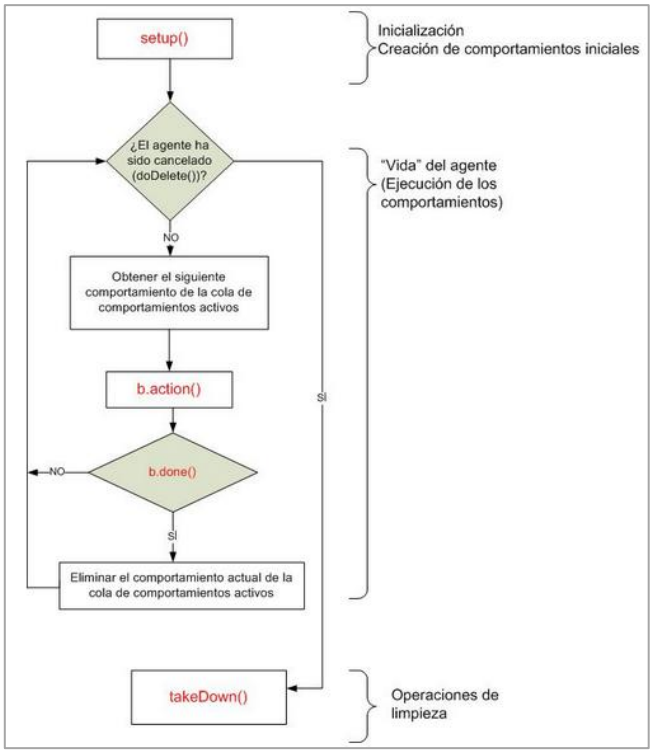

Figura 33 – Flujo de control.

En muchas ocasiones los agentes realizan, o tratan de realizar, funcionalidades complejas que pueden llegar a implicar tareas simultáneas forzando a implementar agentes multihilo, lo que puede causar problemas. JADE proporciona un sistema de comportamientos (*behaviours*) que ayudan al usuario a construir sistemas multiagente y reutilizar código.

El paquete *jade.core.behaviours* contiene las clases que se usan para implementar comportamientos básicos de agentes. Los agentes JADE programan sus comportamientos con un solo hilo y el decidir qué comportamiento se ejecuta en cada momento es tarea del desarrollador del agente. De esta manera se eliminan los problemas de sincronización entre comportamientos concurrentes que acceden al mismo recurso, haciendo que cada agente sea equivalente a un único hilo, con el consiguiente ahorro de ciclos de CPU y memoria. Por tanto, pueden estar activos varios comportamientos a la vez, pero sólo uno de ellos se ejecutará en un momento determinado.

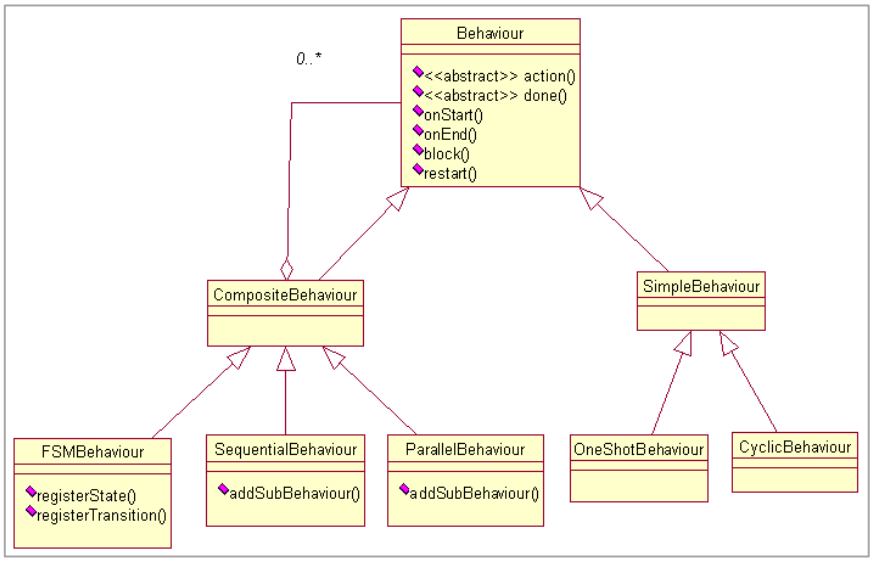

Figura 34 – Diagrama de la clase Behaviour.

Los comportamientos empleados en este trabajo son:

- *OneShotBehaviour*: En este tipo de comportamiento el método *done*() siempre devuelve "*true*", de forma que sólo se ejecuta una vez y de forma ininterrumpida.
- *CyclicBehaviour*: Representa un comportamiento que debe ejecutarse una serie de veces.
  - El método *done*() devuelve false.
  - Se mantiene activo tanto tiempo como esté activo el agente.
  - Hay riesgo de que se pueda quedar con toda la CPU.

3. Comunicación

Los fundamentos de la comunicación en JADE son:

- La comunicación entre agentes es fundamental para poder conseguir la potencia propia de los sistemas multiagente.
- Determina el comportamiento social de los agentes.

- Para que los agentes se puedan comunicar deben usar el mismo lenguaje de comunicación.
- Un lenguaje de comunicación define los tipos de mensaje: informar, solicitar, preguntar, etc.
- Las conversaciones entre agentes se rigen por una serie de protocolos de interacción
- El lenguaje de comunicación de agentes (ACL) permitirá transmitir una serie de conocimiento que vendrá expresado en un lenguaje de contenido. Los términos del lenguaje de contenido que representen conocimiento pertenecerán a un vocabulario común a los distintos agentes que se llama ontología.

En el caso de JADE el lenguaje de comunicación entre agentes es FIPA-ACL. Un mensaje FIPA-ACL puede contener los siguientes campos:

- *performative*: tipo de acto comunicativo (acción que realiza el mensaje). Es el único campo obligatorio y puede tomar uno de los siguientes valores:
  - *accept-proposal*: aceptar una propuesta recibida previamente
  - *agree*: estar de acuerdo en realizar alguna acción
  - *cancel*: cancelar alguna acción pedida previamente
  - *cfp*: solicitar propuestas para realizar una acción dada
  - *confirm*: informar a un receptor que una proposición es cierta
  - *disconfirm*: informar a un receptor que una proposición es falsa
  - *failure*: informar a otro agente que se intentó una acción pero falló
  - *inform*: informar a un receptor que una proposición es cierta
  - *inform-if*: si el agente que recibe la acción cree que la sentencia es verdadera informará de manera afirmativa, sino indicará que es falsa.
  - *inform-ref*: permite que el emisor informe al receptor de un objeto que cree que corresponde a un descriptor, como puede ser un nombre u otra descripción que lo identifique.
  - *not-understood*: informar a un receptor que el emisor no entendió el mensaje
  - *propagate*: el receptor trata el mensaje como si fuese dirigido directamente a él, y debe identificar los agentes en este descriptor y enviarles el mensaje a ellos
  - *propose*: enviar una propuesta para realizar una cierta acción
  - *proxy*: el receptor debe seleccionar agentes objetivo denotados por una descripción dada, y enviarles un mensaje embebido
  - *query-if*: preguntarle a otro agente si una determinada proposición es cierta
  - *query-ref*: preguntar a otro agente por el objeto referenciado en una expresión
  - *refuse*: rechazar realizar una acción
  - *reject-proposal*: rechazar una propuesta durante una negociación
  - *request*: solicitar a un receptor que realice alguna acción
  - *request-when*: solicitar al receptor que realice alguna acción cuando una proposición dada sea cierta
  - *request-whenever*: solicitar al receptor que realice alguna acción cada vez que una proposición dada sea cierta
  - *subscribe*: una intención persistente de notificar al emisor de un determinado valor, y volver a notificarle cada vez que dicho valor cambie

- *sender:* AID del emisor
- *receiver:* lista de AID's de los receptores
- *reply-to:* receptor de un mensaje *reply*
- *content:* contenido del mensaje
- *language:* lenguaje en que se expresa el contenido
- *encoding:* codificación del contenido
- *ontology:* ontología usada para dar significado a los términos del contenido
- *protocol:* identificador del protocolo de interacción
- *conversation-id:* identificador de la conversación. Esto es especialmente útil cuando un agente mantiene varias conversaciones a la vez.
- *reply-with:* indica una expresión que tendrá que ser usada por el agente que responda a dicho mensaje. Si un agente envía un mensaje que contiene *:reply-with query1* el receptor responderá con otro que contenga *:in-reply-to query1*
- *in-reply-to:* Hace referencia a que este mensaje es una respuesta a otro anterior.
- *reply-by:* Indica el tiempo en que el mensaje ha de ser respondido.

En cuanto al envío y recepción de mensajes:

- El intercambio de mensajes entre agentes en JADE se realiza mediante mensajes FIPA-ACL.
- Mecanismo: paso asíncrono de mensajes.
- Cada agente tiene una cola de mensajes entrantes.
- La lectura efectiva de los mensajes es a voluntad del agente.

Un agente puede:

- Leer el primer mensaje en la cola.
- Leer el primer mensaje que satisfaga un requisito.
- La cola de mensajes es única para cada agente y, por lo tanto, es compartida por todos los comportamientos.
- Cada vez que se coloca un mensaje en la cola el agente receptor es avisado.
- Un comportamiento puede ser bloqueado en espera de la recepción de un mensaje: sincronización.
- Los mensajes intercambiados por agentes son instancias de la clase *jade.lang.acl.ACLMessage.*

A continuación se muestran algunos de los métodos más importantes de la clase *ACLMessage*. Para consultar más detenidamente todos los métodos de dicha clase se puede consultar su documentación en la API de JADE (*ACLMessage*).

- setPerformative(int): toma como parámetro una constante representativa de un tipo de acción performativa y la establece como performativa del mensaje. Los posibles actos comunicativos son los que hemos visto antes y las constantes que los representan se pueden ver en la API. Por ejemplo, para hacer que el mensaje *msg* sea de tipo *agree* bastará con escribir: *msg.setPerformative(ACLMessage.AGREE)*;

- *getPerformative():* devuelve un entero equivalente a la constante que representa a la performativa del mensaje;
- *createReply():* crea un mensaje de respuesta para el mensaje sobre el que es aplicado, poniendo los valores oportunos en campos como *receiver*, *conversation-id*, etc.
- *addReceiver(AID):* toma como parámetro un AID y lo añade a la lista de receptores
- *getAllReceiver():* devuelve un iterador sobre la lista de receptores.
- *setContent(String):* recibe como parámetro una cadena y la pone como contenido del mensaje;
- *getContent():* devuelve una cadena con el contenido del mensaje.

El resto de métodos *get/set* son similares, simplemente varía el tipo del valor introducido o devuelto. Por ejemplo, *getSender*() devolverá un AID mientras que *getLanguage*() devolverá una cadena.

La performativa puede indicarse directamente pasando el valor al constructor de *ACLMessage*, lo cual es recomendable porque todos los mensajes deben contener al menos la performativa.

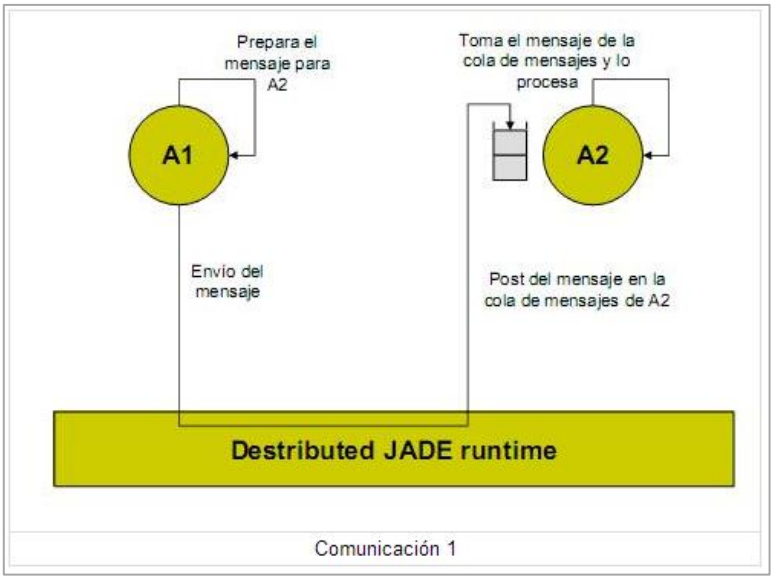

Figura 35 – Proceso de envío y recepción de mensajes en JADE.

4. Ontología

La RAE define el término ontología como la "parte de la metafísica que trata del ser en general y de sus propiedades trascendentales". Sin embargo, en el campo de la informática, "no ha de ser considerada como una entidad natural que se descubre sino como un recurso artificial que se crea" (Mahesh, 1996).

El sinónimo más habitual de ontología es conceptualización:

- Modelo abstracto de algún fenómeno del mundo del que se identifican los conceptos que son relevantes.
- Hace referencia a la necesidad de especificar de forma consciente los distintos conceptos que conforman una ontología.

- Indica que la especificación debe representarse por medio de un lenguaje de representación formalizado.
- Refleja que una ontología debe, en el mejor de los casos, dar cuenta de conocimiento aceptado, como mínimo, por el grupo de personas que deben usarla.

Una Ontología es la "especificación de una conceptualización", la descripción de los conceptos y relaciones entre ellos, que pueden formar parte del conocimiento de un agente o una sociedad de agentes.

La necesidad de utilizar ontologías viene dada por la complejidad inherente a las aplicaciones desarrolladas en el contexto de los Sistemas Multi-Agente, que hace que se presenten las siguientes dificultades:

- Abundancia de comunicación entre agentes.
- Interoperabilidad de sistemas y plataformas.
- Problemas semánticos.

Para que los agentes se comuniquen entre ellos, deben compartir el mismo idioma, vocabulario y protocolos. Al seguir las recomendaciones del estándar FIPA, JADE ya aporta un cierto grado de coincidencia al usar los actos comunicativos FIPA y su lenguaje de contenido SL (*Semantic Language*), que determinan la forma en que los mensajes son intercambiados por los agentes. Sin embargo, será necesario definir ontologías específicas, con su propio vocabulario y semántica del contenido de los mensajes intercambiados por los agentes.

JADE proporciona tres formas distintas de llevar a cabo la comunicación entre agentes:

1. La forma más básica es utilizar cadenas para representar el contenido de los mensajes. Esto es conveniente cuando el contenido de los mensajes son datos atómicos pero no en el caso de conceptos abstractos, objetos o datos estructurados, ya que en estos casos, se tendrían que realizar parseos sobre las cadenas para acceder a sus diversas partes.
2. Otra forma es utilizar objetos serializables de Java, que transmitirían directamente el contenido de los mensajes. Este es el método más conveniente para aplicaciones locales donde todos los agentes son implementados en Java. Un inconveniente es que estos mensajes no son entendibles para las personas.
3. El tercer método consistiría en definir los objetos que van a ser transferidos como extensión de las clases predefinidas por JADE que pueden codificar/decodificar los mensajes a un formato FIPA estándar. Esto permite que los agentes de JADE puedan interoperar con otros sistemas de agentes.

Una ontología en JADE, se define de forma que los agentes se comuniquen utilizando el tercer método descrito. El soporte JADE para ontologías incluye las clases para trabajar con éstas y con los lenguajes de contenido:

- Los lenguajes de contenido tienen que ver con la representación interna del contenido de los mensajes ACL.
- Las ontologías tienen que ver con la semántica de los mensajes que se intercambian y su chequeo.

Una ontología en JADE es una instancia de la clase *jade.content.onto.Ontology*, en la que se añaden los esquemas que definen la estructura de los tipos de predicados, acciones y conceptos relevantes en el dominio en cuestión. Estos esquemas son instancias de las clases *PredicateSchema*, *AgentActionSchema* y *ConceptSchema*, que se incluyen en el paquete *jade.content.schema*.

Estas clases tienen métodos para poder declarar slots que definen la estructura de cada tipo de predicado, acción y concepto. Como una ontología es básicamente una colección de esquemas que normalmente no varían a lo largo del ciclo de vida de un agente, es recomendable declarar la ontología siguiendo el patrón *singleton*, de modo que sólo se pueda crear un objeto de esa clase. Esto nos permite compartir la misma ontología (y todos los esquemas incluidos) entre todos los agentes que se estén ejecutando en la JVM.

Supongamos una ontología para una tienda de frutas, que podemos utilizar para enviarse ofertas entre agentes.

Podemos identificar los diferentes tipos de esquemas de una ontología:

- **Conceptos**: representan las entidades que forman parte de la ontología.
- **Predicados**: son expresiones que relacionan a los conceptos para decir algo. Son necesarios porque en un mensaje nunca podremos enviar conceptos, sino que tendremos que enviar predicados o acciones.
- **Acciones**: son acciones que pueden llevar a cabo los agentes.

La conversión y las operaciones de chequeo son llevadas a cabo por un objeto que gestiona el contenido, este objeto se denomina *ContentManager* y está incluido en el paquete *jade.content*. Cada agente en jade posee un *ContentManager* al que puede acceder usando el método *getContentManager*(). Esta clase proporciona todo los métodos de transformación de un objeto a un *string* para introducirlo en un slot de un *ACLMessage* y viceversa.

La conversión la realiza a partir de:

- Una ontología (la que hemos definido).
- Un códec de un lenguaje de contenido (instancia de la clase *Codec* del paquete *jade.content.lang*).

La ontología permite validar la información desde un punto de vista semántico y el códec realiza la conversión a string según las reglas sintácticas del lenguaje de contenido.

JADE define dos lenguajes de contenidos predefinidos. En el paquete *jade.content* se incluyen dos códec para dos lenguajes de contenido, los tipos de lenguaje son los siguientes:

- Lenguaje SL: legible para las personas y codifica las expresiones como *string* (*jade.content.lang.sl*).
- Lenguaje LEAP: no legible y codifica en *byte-encoded(jade.content.lang.leap*).